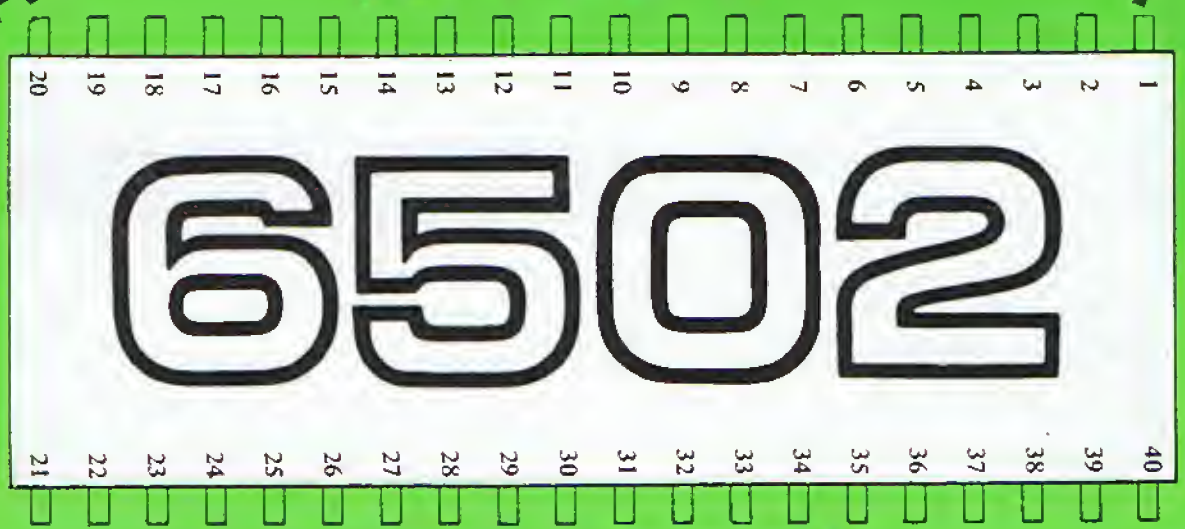
The BEST of MICRO™

AIM 65

KIM - 1

APPLE II



SUPERBOARD

PET

SYM - 1

Volume 2

The BEST of MICRO™

Volume 2

Oct/Nov 78 to May 79

The BEST of MICRO

P.O. Box 6502
Chelmsford, MA 01824
617/256-5515

MICRO is a publication devoted to the world of the 6502 microprocessor: the 6502 based microcomputers, peripheral hardware, software, ideas, applications, and so forth.

MICRO began publication with the Oct/Nov 1977 issue and was published regularly on a bimonthly basis for the first year. During the second year, MICRO became a monthly publication. This volume, "**The Best of MICRO — Volume 2**", contains all of the significant material from issues 7 through 12 of MICRO. Only the advertising, a few minor articles, and a few dated articles have been omitted. Any errors which were discovered after the initial publication of the articles have been corrected in this collection.

MICRO obtains most of its material from its readers: users of 6502 based systems — hobbists and professionals alike. Authors are paid a fee for articles which appear in MICRO, and will obtain additional royalties for reprinting such as this collection.

MICRO is interested in promoting the use of the 6502 and feels that this can best be accomplished by presenting material that is of a useful, informative nature as opposed to lots of games or vague "think" pieces.

MICRO has, in the period Oct/Nov 1978 through May 1979 which is covered in this volume, focused primarily on the KIM, PET, and APPLE microcomputers. This is because the material we recieved was about these three systems. We would welcome material about the OSI systems, or any of the myriad of other 6502 based systems which are not as popular. We also anticipate broad coverage of the new 6502 systems that are just becoming available at the end of the period: the SYM-1 and the AIM-65.

MICRO covers all of the 6502 systems because we feel that ideas generated on one system may often be useful to users of other related systems. Therefore, do not just read the stuff in the section on your particular machine, but find out about the other machines as well, and see what you can adapt to your own uses.

MICRO is now published monthly by MICRO INK, Inc. For information on subscriptions and back issues, write to:

MICRO
P.O. Box 6502
Chelmsford, MA 01824
USA

Editor/Publisher
Robert M. Tripp

CONTENTS

AIM

SYM

KIM

ASK THE DOCTOR — PART I

Robert M. Tripp, Ph. D.
The COMPUTERIST, Inc.
P.O. Box 3
S. Chelmsford, MA 01824

The Rockwell International AIM 65, the Synertek SYM-1 and the Commodore KIM-1 form a closely knit family of microcomputers. Of course they all use the 6502 microprocessor, but the family resemblence is much deeper than that. A few of the features that make the three boards so similar are:

1. Each is a "bare" single board microcomputer without a case, built-in power supply, etc.

2. They have the same basic I/O support:
   A. 20 mA current loop TTY interface; and,
   B. Low Speed Audio Cassette interface. All three computers support the KIM-1 cassette tape format. This means that a cassette tape generated in the KIM-mode on any of the machines can be read on any other machine. This tape cassette compatibility is so complete that it is possible to directly interconnect a KIM to SYM, or KIM to AIM, or SYM to AIM via the the audio cassette interface - without the cassette! Simply take the **Audio Out HI** from one computer and connect it to the **Audio IN** of the other. Then run the Load KIM format cassette program on the second computer and the Write KIM format cassette program on the first computer.

3. They have a compatible bus structure. Each computer has two dual 22 pin edge connectors with essentially the same connections. The **Expansion connectors** have identical placement of all the **Address, Data, Control** and **Power** lines. The **Application connectors** have identical placement of most signals that are common on the three computers - **Port A** and **Port B I/O, Power** and **Ground, Audio Cassette I/O, TTY I/O** - plus some additional signals which are unique to each computer. This bus similarity is a very important component of the AIM/SYM/KIM (**ASK**) family compatibility.

4. The SYM intentionally "duplicates" many of the KIM Monitor routines, and has a similar **Hex Keypad** and **LED Display** on board. The reader is hereby warned to be careful when using SYM routines which proport to be 'the same as" the KIM routines. As will be shown in a later column, there are often minor, but important differences between two routines which at first appear identical. For example, in the KIM **PACKT** subroutine, a successful return is signaled by the **Zero Flag** being **Set**; an error return by the **Zero Flag** being **Cleared**. The similar SYM **PACKT** subroutine performs the same packing function, but signals a successful return with the **Carry** bit **Cleared**; an error return by the **Carry** bit **Set**. So, be careful.

An AIM/SYM/KIM Compatibility Example

One way to understand the nature of the similarities and differences between the ASK family members is to examine in detail a common situation which involves both hardware and software for the three systems. **MEMORY PLUS**(tm) is a multi-purpose board that was designed for the KIM-1 long before the SYM or AIM were even a gleam in their creators' eyes. It contains **8K RAM**, provision for up to **8K EPROM,** a **6522 Versatile Interface Adapter**, and an **EPROM Programmer**. Since it was designed to work on the KIM-1, it obviously is compatible with that computer. The question is: Is the MEMORY PLUS compatible with the SYM and AIM? The answer is Yes, No, and Maybe. Let's examine this seeming paradox in some detail.

YES

The **8K RAM** and the **8K EPROM** work directly with the KIM, SYM and AIM with no modification. In fact, the same connector cable may be used to connect the MEMORY PLUS to any one of the computers. This exact compatibility is due to the fact that all that MEMORY PLUS requires for operating the **RAM** and **EPROM** are the **Address, Data, Control** and **Power** lines, and these are all positioned identically on the **Expansion connector.**

NO

The addressing of the **6522 VIA I/O** was designed to use the **K5** chip select that is generated by the KIM and which appears on the **Application connector**. This same signal is generated by the SYM and makes the addressing of the **6522 VIA** identical to that of the KIM. The AIM does not generate this signal. Therefore, without some sort of modification, the AIM can not use the **6522 VIA**, and since this is the heart of the **EPROM Programmer**, can not program **EPROMs**. Fortunately, there are a couple of unused gates on the MEMORY PLUS and a minor wiring modification can be made so that the MEMORY PLUS will itself generate the equivalent of the **K5** signal and permit the AIM to use the **6522 VIA** and **EPROM Programmer.** This does point out a small, but significant difference, between the bus signals of the KIM, SYM and AIM. In general, the SYM made much more of an effort to be KIM compatible than the AIM did. This example where the KIM and SYM generate the **K1, K2, K3, K4,** and **K5** signals and the AIM does not, is probably the greatest difference in the hardware as seen on the **Application** and **Expansion busses.**

MAYBE

Since the KIM does not do all of the address decoding required for a system beyond the initial 8K used by the KIM on board, any additional memory device must generate a **DECODE** signal which enables the KIM memory at the proper times. The MEMORY PLUS board has circuitry to generate the **DECODE**. The SYM and the AIM do all of the required address decoding for their operation on-board, and do not therefore require this signal. The **DECODE** signal may be simply ignored in these two systems by not connecting it from the MEMORY PLUS to the SYM or AIM.

There are other addressing space differences between the three systems, which may or may not be important in a particular

situation. All three have **RAM** in locations 0000 to 03FF. This includes the **Page Zero** and **Stack** locations. The KIM does not use 0400 to 16FF, but uses 1700 to 177F for **I/O** and **Timers**, 1780 to 17FF for **RAM**, and 1800 to 1FFF for the **ROM Monitor**. The AIM has 0400 to 0FFF available for on-board **RAM** expansion, 1000 to 9FFF are available for **User** expansion, A000 to AFFF is used for **I/O** and **System RAM**, and the remainder of the memory is allocated for various ROMs: B000 to CFFF for BASIC, D000 to DFFF for Assembler, and E000 to FFFF for Monitor. The SYM has 0400 to OFFF for on board **RAM** expansion, 1000 to 7FFF for **User** expansion, 8000 to 8FFF for **Monitor ROM**, 9000 to 9FFF reserved for **Monitor** expansion, A000 to AFFF for **System RAM** and **I/O**, B000 to BFFF for **User** expansion, C000 to DFFF for **BASIC ROM**, E000 to FF7F reserved for **Assembler/Editor ROM**, and FF80 to FFFF for **SYSTEM RAM Echo** locations. The above listing of memory allocation should make it obvious that the three systems each have **I/O** and **Monitors** located in different places, so that software calling on the **I/O** or **Monitor** will have to be at least different in the addresses used. On the MEMORY PLUS this shows up when the host computer's **Port B** is used to generate three of the addresses required by the **EPROM Programmer**. While the three lines, **PB0, PB1,** and **PB2** are all mapped to the same Application connector locations, the address of the **I/O** device controlling the port is different. In fact, the **I/O** device on the KIM is a **6530** and the device on the SYM and AIM is a **6522**! All this does is require different addresses within the EPROM Programming program. Another memory mapping difference is in the location of the interrupt vectors. Each of the three computers uses different addresses to handle the interrupts. The MEMORY PLUS programmer uses the **IRQ interrupt**, and must therefore set up the **IRO vector** in a different location on the KIM, SYM or AIM. Again, this is a minor problem, but is an incompatibility. Finally, since the Monitor is in a different location in each computer, a return to the Monitor at the end of the **EPROM program** will be to a different address for each. If the MEMORY PLUS used the on-board **Timers**, then it would again require some modifications to the software. In the case of the KIM, the **Timer** is of the **6530** variety; the SYM and AIM have **6522** types. This would require a different set of parameters as well as different addresses. As a matter of fact, MEMORY PLUS uses its own **6522 Timer**, and so this problem does not arise.

One final note of caution on the memory allocation of the three computers. Even though they all support **RAM** in locations 0000 to 03FF, the use of this **RAM**, especially the end of **Page Zero**, is quite different between them, both in the amount of **Page Zero RAM** used and the use of particular locations. In addition, while the KIM and the SYM do not use **Page One** for anything, in general, except as the **Stack**, the AIM makes extensive use of **Page One**. This variation in use of **Page Zero** and **Page One** will often require that existing programs undergo some re-definition of addresses and a re-assembly before they can be moved from one computer to another, even when the **Monitor** of the computer is not being used as part of the program.

SUMMARY

The AIM/SYM/KIM family of 6502 based microcomputers have a lot in common; but they also have some significant differences. In most cases these differences are not so great that they can not be overcome with some careful modification to existing hardware and/or software. But, significant differences do exist, and any user who plans to use a variety of these systems should be aware of the potential problems that exist. Subsequent columns will go into more detail on the similarities and differences between the ASK family members.

SYM Cassette Tape Problems

There are two problems with the SYM tape service that users should be aware of. The first is that the SYM hardware has a filter circuit that is used in shaping the input signal from the cassette recorder. This particular circuit is very sensitive and will not work reliably with all tape recorders. It apparently was optimized to a particular type of unit, possibly a SuperScope C-190; and is not very optimal for a large number of other units. Several suggestions have been made to improve this circuit. One is to replace the resistor R92 (see page 4-9 in the SYM Reference Manual for a circuit diagram) which is a 1K with a 3.3K. Another idea that has been used was to put a .01 MFD capacitor in parallel with C15 which is a .47 MFD. I have **NOT** had a chance to try either of these and do not guarantee that they either work or that they will not destroy your system. I am merely passing on a couple of suggestions which were given to me. I hope to be able to give a more complete and tested set of changes by next month.

The second tape problem has to do with reading KIM format tapes. As you probably know, the KIM format uses an ASCII "/" character to signal the end of data. This character has a hex value of 2F. The SYM Monitor has software to detect the end of data character which properly detects an ASCII "/" as it should. However, it also has software which erroneously thinks that an ASCII "2" followed by an ASCII "F" which when combined make a hex 2F data byte, is a terminator. This means that anytime your data has a 2F in it, as in

```
4C 13 2F  JMP $2F13  (Jump to address 2F13)
```

it will mistake the legitimate 2F data as a "/" character and think that it has reached the end of the data. Since the following bytes of data will be considered to be the check digits, and will not be correct, the SYM will give you an error and stop loading. This can be very disheartening. Synertek is aware of the problem and is supposed to fix it, but no fix has been received here yet.

One way I have overcome this difficulty, with some difficulty, is to load my program into the KIM, change any 2F data to an FF, and then either make a cassette tape or dump the data directly into the SYM from the KIM via the Audio Out HI on the KIM to the Audio IN on the SYM. Then I have to go to the SYM and change all of the FF's which were substituted for the 2F's back to their original 2F value. This is cludgy, but it works. If you do not have a KIM handy, however, you are out of luck.

Coming Attractions

Future columns will cover all sorts of interesting information about the AIM, SYM, KIM (and maybe SUPERKIM). If you have discovered any useful bits of information about these machines, please drop me a line and I will try to include the info in a future column. In this way the material can be widely disseminated without your having to write a whole article about it.

Note: MEMORY PLUS(tm) is manufactured by The COMPUTERIST, Inc., P.O. Box 3, S. Chelmsford, MA 01824. It currently retails for **$200**00

ASK THE DOCTOR — PART II
AN ASK EPROM PROGRAMMER

Robert M. Tripp, Ph. D.
The COMPUTERIST, Inc.
P.O. Box 3
So. Chelmsford, MA 01824

One of the most frequently asked questions about the **ASK** (AIM/SYM/KIM) family of microcomputers is: "Can a program that was written for one of the micros run on either of the others?" The answer is normally no. While the three micros share a lot - common expansion bus, similar application connector, KIM tape format ... they do have minor differences in their use of page zero and page one, some greater differences in their memory and I/O allocations, and large differences in their monitor subroutines. Therefore, in general, the answer to the question is: "No, a program written to run on one will not run on the others without modification." This answer may lead the creative programmer to wonder what it would take to write programs which would run on all three machines, without requiring customization for each. What problems would be encountered? What techniques could be used to reduce the problems? What about ...?

I faced the three-machine problem for a practical reason. the MEMORY PLUS™ board that my company makes is hardware compatible on the three systems. Part of the package is a cassette tape with a Memory Test program and an EPROM Programming program. It would be awkward to have to provide three sets of programs on the tape and expensive to have to print up three different sets of program listings. Would it be feasible to write a single program? The answer turned out to be: "Yes". The program for the EPROM Programmer is presented here in its entirity.

There are two major types of compatibility problems. The first is that the three monitors each have a different set of support subroutines. Sometimes they may have identical subroutines, but usually the subroutines are not identical, and often are not even close! In this particular program, this was not a problem since the program did not use any monitor subroutines. The second major problem is that various important locations in memory or in memory mapped I/O are different on the three systems. Examples are the re-entry address for returning to the monitor at the end of the program, the location of the interrupt vector, and the address of the peripheral I/O port. In this program all three of these address problems were encountered. The solution for the addressing problem is fairly simple and will handle all three addressing problems - if you understand the **Indirect Indexed** mode of addressing on the 6502. If you are totally unfamiliar with this addressing mode, you should consult your programming manual at this point and find out about it. If you are familiar with it, then this review may be useful.

The **Indirect Indexed** addressing mode on the 6502 works by having a base pointer in a pair of page zero locations which is used to point to some other location in memory. The contents of the page zero locations are combined with current contents of the Y register to form the final address for an instruction. The assembler form of the instruction is LDA (POINT), Y in the standard MOS Technology syntax or LDAIY POINT in the MICRO-ADE syntax which is generally used in MICRO. In either case, what results is a form of addressing in which the page zero pointer forms the base address and the contents of the Y register allow this address to be modified within a range of 00 to FF. If the pointer value was 2800, then the effective range of the indirect indexed instruction would be 2800 (with Y = 00) to 28FF (with Y = FF). The page zero pointer is set up in two consecutive bytes, with the low byte of the address first followed by the high byte of the address. In our example, if POINT was the page zero address 0006, then location 0006 would contain 00 (the low byte of the indirect address) and 0007 would contain 28 (the high byte of the indirect address). Since the only problem we have to solve for the EPROM Programmer is one of different addresses for the three systems, the problem reduces to three steps:

1. Determine which system we are runnng on: AIM, SYM or KIM.
2. Set up appropriate indirect address pointers.
3. Access the variable addresses via the indirect address pointers using the Indirect Indexed addressing mode.

Now Let's examine the program in a little detail to see how it actually accomplishes all of this.

The Program

The program is assembled to run entirely on page zero. It uses a 6522 VIA chip which is located on the MEMORY PLUS board for a lot of its I/O and timing. The registers within the VIA that are used are listed under VIA REGISTER OFFSETS. These offsets will be used within the program to load the Y register prior to making an Indirect Indexed instruction call so that the appropriate VIA internal register will be accessed. The first six locations in page zero are used by the program for parameters to control where the data to be placed into the EPROM starts in memory, ends in memory, and where it is to be placed in the EPROM. This information is filled in by the operator before running the program. Location **"VIA"** is an indirect pointer to the MEMORY PLUS VIA chip. This normally will be at location 6200 and could have been addressed directly by the program. But, since it could be in another address, it was decided to handle it through the Indirect Indexed mode. The **"JMPMON"** location contains the Opcode for a JMP. This is used in conjunction with the contents of the next two bytes, **"MONTOR"**, to re-enter the system monitor at the end of the program or when an error is encountered. The actual monitor re-entry address value is filled in by the program. It appears as 0000 in the listing, but will be altered early in the program as we shall see below. The **"INTVEC"** is an indirect pointer to the IRQ interrupt vector which is used as part of the timing service of the program. This will be properly filled in at the beginning of the program from a table. **"PBDD"** and **"PBD"** are pointers to the Port B Data Direction and Port B Data registers. These will also be filled in from a table at the start of the program and will be used in Indirect Indexed instructions.

The program begins execution at location 0011, after the user has used his monitor to fill in the appropriate values in the parameters in locations 0000 to 0005. The first three instructions clear all of the status bits by pushing a 00 onto the stack from A and popping it into the status register.

Locations 0015 through 0027 determine which microcomputer the program is running on by testing the contents of a ROM location. The contents of location FFFD is specific to each machine. This is the high order byte of the Reset Interrupt Vector. For the SYM this will be an 8B; for the AIM an E0, and for the KIM a 1C. The X register is loaded with a value which is the start of a table of values which will be moved into locations 0009 through 0010 to fill in the **MONTOR, INTVEC, PBDD,** and **PBD** pointers discussed above. The instruction at 0028 is unique to the SYM and is required to permit the program to access some of the SYM's protected memory locations. It is not executed by the program for KIM or AIM.

Locations 002B through 0035 move the appropriate table from its original location at the end of the program into the working indirect area. The AIM table starts at 00D0; the KIM table at 00D8; the SYM at 00E0.

By the time we reach **ENTER** at location 0036, two important things have been done. First, we have determined which machine we are running on. Second, using this information, we have set up our indirect pointers which will be used by the remainder of the program to address the machine specific addresses. At **ENTER** we again set the status bits to zero. This is done so that a user with a different computer could still use this program. He would do this by manually setting up the pointers in 0009 through 0010 and then starting at 0036 - **ENTER.**

Locations 003A through 0044 fill in the system interrupt vector to point to the interrupt servicing routine of the program which starts at 00C5. This is a good place to examine the workings of the Indirect Indexed addressing. The Y register is set to 00. The A register is loaded with the low byte of the interrupt service routine address. This value will be C5 since the routine starts at 00C5. This is then stored in the system interrupt vector which is addressed by adding the contents of Y (00) to the address contained in **INTVEC.** For the AIM **INTVEC** will have been set to A400; for the KIM **INTVEC** will be 17FE; for the SYM A67E. So the effective address will be A400 for the AIM (A400 + 00= A400), 17FE for the KIM and A67E for the SYM. The A register is then loaded with the high byte of the interrupt service routine address, 00 since the routine is in page zero. The Y register is incremented so that it now contains a 01. When A is now stored with Indirect Indexed mode through **INTVEC,** it goes into A401 on the AIM (A400 + 01 = A401), 17FF on the KIM and A67F on the SYM. If you are not clear at this point as to how this works, then **STOP.** The rest of this article will make no sense until you understand the basics of the Indirect Indexed mode. Re-read the article to this point, consult your manual, ask a friend.

Using the same techniques of setting Y to an offset value, loading A with the value to use, and storing in the Indirect Indexed mode, the VIA is initialized.

The instructions from 005D through 0078 set up the VIA for output. One additional trick is used here. While we normally think of the Y register in connection with the Indirect Indexed mode of addressing, the X register can also be used for this mode of addressing - but only under one special condition. That condition is when the index value is 00. In this condition, the Indirect Indexed mode and the Indexed Indirect mode both collapse to the simple Indirect mode. There are several places in which we take advantage of this fact so that the X register can be set to zero once and used several times for addressing. This section of code now gets the data from the indirect pointers that the operator set into locations 0000 through 0005 and outputs the data to the EPROM Programmer.

Locations 0079 through 008A first set a timer in the VIA going for the 50 millisecond period which is required to program one location on the EPROM. Then the Peripheral Control Register on the VIA is set to enable the programming pulse to the EPROM. Again, Indirect Indexed addressing is used so that the VIA does not have to be at 6200. If it is in any other address, the operator simply sets the pointer at VIA (0006, 0007) before starting the program. Everything else is automatic.

Locations 008B to 008E form a loop which waits until an interrupt has occurred and been serviced. If you look down at the interrupt routine starting at 00C5 you will see that Y is changed so that it is no longer equal to 0C. At this point the **WAIT** test will fail and the program will move on to **VERIFY.**

Locations 008F through 00C4 perform a series of tests and pointer updates. When the program reaches the end of the data, or if it detects an error, it makes a **JSR** to **JMPMON. JMPMON** then jumps to a re-entry point for the appropriate monitor as set up from the table at the beginning of the program. The reason for making the JSR is to save the address of where we are coming from to be displayed by the monitor as an indication of why we exited: successful completion or one of the three errors. The **JMPMON** permits us to go to the correct monitor. While it would have been possible to have the initialization code change each of the four JSR's to JSR directly to the appropriate monitor, this obviously would have entailed more code and would not have any benefit.

The re-entry to the monitor is the only place where this code makes use of the system monitor, and wouldn't you know it - each monitor handles the re-entry slightly differently. They each display an address which is related to the JSR from which it came, but each one displays a slightly different address. On the successful completion return which is at 00B7, the AIM displays 00B8, the KIM displays 00B9, and the SYM displays 00BA. It would have been possible to write some additional code to take care of the address before returning to the monitor, but this did not seem to be a serious enough problem to warrant the effort. But it does point out the problems one can encounter in using the "similar-but-different" monitor subroutines.

Locations 00C5 through 00CF are the interrupt service. When the interrupt occurs, it is vectored here due to the setup that took place earlier in the program. The VIA is changed from programming mode to verify mode and the interrupt is cleared. In the process the Y register is changed so that the **WAIT** test will permit the program to recognize that an interrupt has occurred and to continue.

The **ATABLE, KTABLE** and **STABLE** are the pointer values for the AIM, KIM and SYM respectively. At the start of the program they are moved into a standard set of locations starting at 0009 (**MONTOR**).

```
                    PROM PROGRAMMER  10 FEBRUARY 1979

                    PROM   CRG    $0000

                    ACCESS *      $8B86  SYM-1 ACCESS ENTRY

                    VIA REGISTER OFFSETS

                    ORB    *      $C000  OUTPUT REGISTER B
                    ORA    *      $0001  OUTPUT REGISTER A
                    DDRB   *      $0002  DATA DIRECTION REGISTER B
                    DDRA   *      $0003  DATA DIRECTION REGISTER A
                    TTWOL  *      $0008  TIMER TWO LOW
                    TTWOH  *      $0009  TIMER TWO HIGH
                    PCR    *      $000C  PERIPHERAL CONTROL REGISTER
                    IFR    *      $000D  INTERRUPT FLAG REGISTER
                    IER    *      $000E  INTERRUPT ENABLE REGISTER

0000 00             SAL    =      $00    STARTING ADDRESS LOW
0001 00             SAH    =      $00    STARTING ADDRESS HIGH
0002 00             PRMLOW =      $00    EPROM LOW ADDRESS
0003 00             PRMHGH =      $00    EPROM HIGH ADDRESS
0004 00             EAL    =      $00    END ADDRESS LOW
0005 00             EAH    =      $00    END ADDRESS HIGH
0006 00             VIA    =      $00    POINTER TO VIA
0007 62                    =      $62    NORMALLY AT 6200
0008 4C             JMPMON =      $4C    JUMP TO MONITOR
0009 00             MONTOR =      $00    POINTER TO SYSTEM MONITOR
000A 00                    =      $00    FOR RETURN FROM PROGRAMMER
000B 00             INTVEC =      $00    POINTER TO INTERRUPT VECTOR
000C 0C                    =      $00
000D 0C             PBDD   =      $00    PORT B DATA DIRECTION
000E 00                    =      $00
000F 00             PBD    =      $00    PORT B DATA
0010 00                    =      $0C

0011 A9 00          BEGIN  LDAIM  $00    CLEAR ALL STATUS FLAGS
0013 48                    PHA
0014 28                    PLP
0015 A2 E0                 LDXIM  STABLE ASSUME SYM
0017 AD FD FF              LDA    $FFFD  TEST HIGH BYTE OF INTERRUPT VECTOR
001A C9 8B                 CMPIM  $8B    = 8B FOR SYM-1
001C F0 0A                 BEQ    SYM
001E A2 D0                 LDXIM  ATABLE ASSUME AIM 65
0020 C9 E0                 CMPIM  $EC    = EC FOR AIM 65
0022 F0 07                 BEQ    MOVE   IT IS THE AIM
0024 A2 D8          KIM    LDXIM  KTABLE ASSUME KIM
0026 D0 03                 BNE    MOVE
0028 20 86 8B       SYM    JSR    ACCESS SYM REQUIRES ACCESS

002B 86 30          MOVE   STXZ   TABLE  +01 SETUP POINTER
002D A2 07                 LDXIM  $07    MOVE 8 BYTES
002F B5 00          TABLE  LDAX   $00    REPLACED BY TABLE
0031 95 09                 STAX   MONTOR MOVE TO MONTOR TABLE
0033 CA                    DEX
0034 10 F9                 BPL    TABLE  MOVE UNTIL X = FF
```

```
0036 A9 00     ENTER  LDAIM $00    CLEAR ALL STATUS FLAGS
0038 48               PHA
0039 28               PLP
003A A0 00            LDYIM $00    ENTRY IF TABLE PRESET
003C A9 C5            LDAIM INTRPT GET INTERRUPT POINTER
003E 91 0B            STAIY INTVEC SETUP IN TABLE
0040 A9 00            LDAIM INTRPT /
0042 C8               INY          BUMP POINTER
0043 91 0B            STAIY INTVEC
0045 A9 EC            LDAIM $EC    SETUP VIA VALUES
0047 A0 0C            LDYIM PCR
0049 91 06            STAIY VIA
004B A0 0E            LDYIM IER    DISABLE ALL INTERRUPTS
004D A9 7F            LDAIM $7F
004F 91 06            STAIY VIA
0051 A0 0D            LDYIM IFR
0053 A9 FF            LDAIM $FF    CLEAR INTERRUPT PENDING
0055 91 06            STAIY VIA
0057 A0 0E            LDYIM IER
0059 A9 AC            LDAIM $AC    ENABLE TIMER TWO
005B 91 06            STAIY VIA

005D A2 00     NEXT   LDXIM $00    INIT X REGISTER
005F A9 FF            LDAIM $FF    SET DATA DIRECTION
0061 A0 02            LDYIM DDRB
0063 91 06            STAIY VIA
0065 A0 03            LDYIM DDRA
0C67 91 06            STAIY VIA
0069 81 0D            STAIX PBDD
006B A5 02            LDA   PRMLOW OUTPUT NEXT ADDRESS
006D 81 06            STAIX VIA    LOW 8 BITS
006F A5 03            LDA   PRMHGH
0071 81 0F            STAIX PBD    BITS 8, 9, 10
0073 A1 00            LDAIX SAL    GET DATA BYTE
0075 A0 01            LDYIM ORA
0077 91 06            STAIY VIA    OUTPUT VIA ORA

0079 A9 50     TIMER  LDAIM $50    SETUP 50 MILLISECOND TIMER
007B A0 08            LDYIM TTWOL
007D 91 06            STAIY VIA    OUTPUT TO TIMER TWO LOW
007F A9 C3            LDAIM $C3    HIGH BYTE OF TIMER
0081 A0 09            LDYIM TTWOH
0083 91 06            STAIY VIA    OUTPUT TO TIMER TWO HIGH
0085 A9 CE            LDAIM $CE    PROGRAM HIGH, PROGRAM MODE
0087 A0 0C            LDYIM PCR
0089 91 06            STAIY VIA

008B C0 0C     WAIT   CPYIM PCR    TEST FOR INTERRUPT SERVICED
008D FC FC            BEQ   WAIT   ELSE, WAIT FOR IT

0C8F A9 00     VERIFY LDAIM $00    VERIFY PROGRAMMING
0091 A0 03            LDYIM DDRA   SET ORA FOR INPUT
0093 91 06            STAIY VIA
0095 A0 01            LDYIM ORA    SETUP POINTER
0097 B1 06            LDAIY VIA
```

```
0099 C1 00          CMPIX SAL    COMPARE ORIGINAL DATA
009B F0 03          BEQ   OKAY   GOOD IF MATCH
009D 20 08 0C       JSR   JMPMON EXIT ON ERROR

00A0 E6 00     OKAY INC   SAL    BUMP DATA POINTER
00A2 D0 07          BNE   TEST   BRANCH IF NOT ZERO
00A4 E6 01          INC   SAH    BUMP HIGH DATA POINTER
00A6 D0 03          BNE   TEST   BRANCH IF NOT ZERO
00A8 20 08 00       JSR   JMPMON EXIT ON ERROR

00AB A5 05     TEST LDA   EAH    TEST ALL DONE
00AD C5 01          CMP   SAH    BY COMPARING POINTERS
00AF D0 09          BNE   MCRE
00B1 A5 04          LDA   EAL
00B3 C5 00          CMP   SAL
00B5 D0 03          BNE   MCRE
00B7 20 08 00       JSR   JMPMON DONE.

00BA E6 02     MCRE INC   PRMLCW BUMP PROM POINTERS
00BC D0 9F          BNE   NEXT   READY IF NOT ZERO
00BE E6 03          INC   PRMHGH BUMP HIGH POINTER
00C0 D0 9B          BNE   NEXT   OKAY IF NOT ZERO
00C2 20 08 00       JSR   JMPMON EXIT ON ERROR

00C5 A9 EC     INTRPT LDAIM $EC    RESET PROGRAM LOW, VERIFY MODE
00C7 91 06          STAIY VIA
00C9 A0 0D          LDYIM IFR    SETUP TO CLEAR INTERRUPT
00CB B1 06          LDAIY VIA    READ AND WRITE TO CLEAR
00CD 91 06          STAIY VIA    INTERRUPT VIA SNEAKY TRICK
00CF 40             RTI          RETURN FROM INTERRUPT

00D0 6D        ATABLE =     $6D    AIM 65 MONITOR ENTRY
00D1 E1               =     $E1    TO DISPLAY PC COUNTER
00D2 0C               =     $00    IRQ INTERRUPT VECTOR
00D3 A4               =     $A4
00D4 00               =     $00    PBDD
00D5 A0               =     $A0
00D6 02               =     $02    PBD
00D7 AC               =     $A0

00D8 C5        KTABLE =     $05    KIM MONITOR ENTRY
00D9 1C               =     $1C
00DA FE               =     $FE    IRQ INTERRUPT POINTER
00DB 17               =     $17
00DC 03               =     $03    PBDD
00DD 17               =     $17
00DE 02               =     $02    PBD
00DF 17               =     $17

00E0 35        STABLE =     $35    SYM ENTRY POINT
00E1 80               =     $80
00E2 7E               =     $7E    IRQ INTERRUPT POINTER
00E3 A6               =     $A6
00E4 00               =     $00    PBDD
00E5 A0               =     $AC
00E6 02               =     $02    PBD
00E7 AC               =     $A0
```

ASK THE DOCTOR - PART III
BITS AND BYTES

Robert M. Tripp, Ph.D.
The COMPUTERIST, Inc.
P.O. Box 3
So. Chelmsford, MA 01824

The Doctor was busy this month and did not get a chance to write up the EPROM Programmer hardware as promised in the last issue. Look for it next time. A couple of people did submit some good info which is printed below. The Doctor encourages such input. Too much is happening with these new computers for anyone person to "know it all", so if you find out something interestng, please drop us a note and let us get the word out.

Corrected AIM SYNC Program

The early AIM User Manuals had a number of mistakes, as is to be expected the first batch. One of the more serious errors was in the listing for the SYN Write and SYN Read programs on page 9-11. The errors have been corrected in later versions of the manual, but for those of you who need the programs, here they are - corrected.

SYN Write Program:

```
0300  20 1D F2   JSR F21D
0303  20 4A F2   JSR F24A
0306  4C 03 03   JMP 0303
```

SYN Read Program:

```
0310  A2 00      LDX #00
0312  A9 CE      LDA #CE
0314  20 7B EF   JSR EF7B
0317  20 EA ED   JSR EDEA
031A  A2 00      LDX #00
031C  A9 D9      LDA #D9
031E  20 7B EF   JSR EF7B
0321  20 29 EE   JSR EE29
0324  C9 16      CMP #16
0326  F0 F9      BEQ 0321
0328  D0 E6      BNE 0310
```

Patch for the AIM-DISASSEMBLER

It soon becomes obvious, that the disassembler is extremely paper consuming, because no single-stepping is provided. The following program will save you money and time!

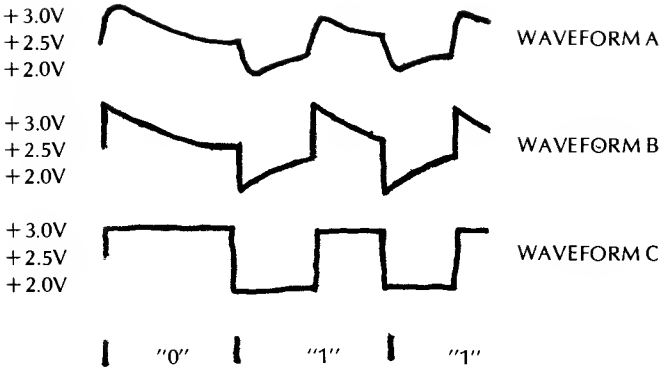
Set F1 (010C) to 'JMP 03D9' and F2 (010F) to 'JMP 03CB'. After loading the desired program address (*), hitting **F1** will dissable just this line on the display. To advance, press the space-bar. If you want to modify, use 'I' and the program jumps to the **Instruction Mnemonic Entry.** The current address will not be changed. **'ESC'** brings you back to the AIM—Monitor. With **'F1'**, the next address will be disassembled. **'F2'**, however, will substract the last used op-code length from the current address and then disassemble the last entry! It is even possible to disassemble further "backwards", just keep switching from **'ESC'** to **'F2'**. Of course, a change in the op-code length will bring up some unexpected results, but very soon you'll catch a proper op-code again!

```
03CB  AD 25 A4   LDA A425
03CE  18         CLC
03CF  E5 EA      SBC EA
03D1  8D 25 A4   STA A425
03D4  B0 03      BCS 03D9
03D6  CE 26 A4   DEC A426
03D9  20 24 EA   JSR EA24
03DC  20 6C F4   JSR F46C
03DF  20 07 E9   JSR E907
03E2  20 3C E9   JSR E93C
03E5  C9 49      CMP #49
03E7  D0 03      BNE 03EC
03E9  4C 9E FB   JMP FB9E
03EC  C9 20      CMP #20
03EE  D0 F2      BNE 03E2
03F0  AD 25 A4   LDA A425
03F3  38         SEC
03F4  65 EA      ADC EA
03F6  8D 25 A4   STA A425
03F9  90 DE      BCC 03D9
03FB  EE 26 A4   INC A426
03FE  90 D9      BCC 03D9
```

Submitted by
Gebhard Brinkmann
Koblenzer Str. 1.
D-5401 Kaltengers
West Germany

SYM Tape Evaluation

As a result of our telephone conversation on Monday, I decided to look for any possible hardware problems in the SYM Cassette Interface. Some results are shown below. Whether these are related to your cassette problems is unknown. In checking my Sony TC-62, I found an unexpected very slow acting AVC (increases gain very slowly, decreases rapidly). This could cause problems in a level sensitive system as the gain slowly increases during the recording process to a quite large degree.



740 usec
wide

All waveforms taken at PIN 3 of the LM311 (U26) with a sync tape generation program running (hi-speed). Audio OUt (HI) is connected directly to Audio In (A-P to A-L).

WAVEFORM A is the normal condition as received (VIM 80650912 E/C0003)
WAVEFORM B is with C14 (.0047uF) removed
WAVEFORM C is with C14 removed and C16(.01uF) paralled with 1uF

CONCLUSION: C16 is much too small and could easily cause the system to become marginal in the presence of noise and normal level variations. C14 has no apparent real value and seems to unnecessarily increase transition time uncertainty. The small value of C16 and the presence of C14 together simulate the waveform degradation of a very limited bandwidth recorder. Their effect augment rather than compensate for the deficiencies of a recorder. Suprisingly, it appears that it would be a recorder with poor low, rather than high, frequency response which would be most likely to have problems with C16 is maintained at its original .01 microfarad value.

Submitted by
Don Lloyd
101 Western Ave., Apt. 76
Cambridge, Ma. 02139

Comments on Synertek BASIC (8K) V1.1

1) 2 ROM's, U21, U22, C000-DFFF, (J) (0) (CR to start BASIC
2) **Commands** - CLEAR, LIST, NULL, RUNN, NEW CONT, LOAD "A", SAVE "A"
3) **Statements** - DATA, DEF, DIM, END FOR, GOTO, GOSUM, IF...GOTO, IF...THEN, INPUT, LET, NEXT, ON... GOSUM, POKE, PEEK, PRINT, READ, REM, RESTORE, RETURN, STOP, WAIT.
4) **Functions** - ABS(X), INT(X), RND(X), SGN(X), SQR(X), TAB(I), USR(I), USR(I,J,...Z), EXP(X), FRE(X), LOG(X), POS(I), SPC(I)
SIN(X), COS(X), TAN(X), ATN(X) all must be loaded separately - App Note 53-SSC not quite available.
5) **Strings** - DIM A$, LET A$, INPUT X$, READ X$, PRINT X$
6) **String Functions** - ASC(X$), CHR$(I), FRE(X$), LEFT$(X$,I) LEN(X$), MID$(X$,I), MID$(X$,I,J), RIGHT$ (X$,I), STR$(X), VAL(X$)
7) **Operators** =,-,+, exponentiation, *,=, (not equal), , , (LTE), (GTE), NOT, AND, OR
8) **Uses Memory** from 0200 HEX up until ROM or no memory, unless restricted at start up.
9) **Weaknesses** - Only editing is delete line, delete last character (RUB-OUT), no ROM TRIG, no program merging capability.
10) **Strengths** - Good array features (but no MAT functions), 9 digit accuracy floating points
4 byte floating point numbers
7 bits + 1 bit sign exponent
1 bit sign +24 bit binary value (**M$**bit = 1 always)
& "000F" = 15 decimal
hex string conversion to decimal
USR (I,J,...Z) Machine language subroutine multiple parameters on stack result (A,Y)

Speed is comparable to OSI Kilobaud Oct '77 ratings (1MHz) Overall subjective by infrequent BASIC user: 7.5/10 seems appropriate to overall product.

Submitted by
Don Lloyd
101 Western Ave., Apt. 76
Cambridge, Ma. 02139

NOTE: Since this article was originally published, Synertek Systems **has released the new** SYSMON V1.1. **It is available through your** SYM **dealer for $15.00 and may be used to update your old version system. New** SYM-1 **systems will come with the new monitor in place.**

ASK THE DOCTOR - PART IV
GOOD NEWS/BAD NEWS

Robert M. Tripp, Ph.D.
The COMPUTERIST, Inc.
P.O. Box 3
So. Chelmsford, MA 01824

In last month's issue I announced that Synertek Systems has informed me of an improvement to the SYM monitor which should solve the audio cassette sensitivity problem that I had mentioned in several columns. I have since received a copy of the new SYM-1 Supermon Version 1.1 on a pair of EPROMs (which I had supplied to them) and have had some chance to evaluate the new version. The documentation I received was in the form of a two page letter. Not having the monitor listing limited by ability to fully evaluate the changes.

The Good News

According to the letter only two minor hardware changes are required in the cassette circuit. This are similar to some reported indepndtly by other users and reported in an earlier column. "Change C16 to .22 microfarad" and "change R97 to 1K ohm".

This list of improvements that accompanied the V1.1 monitor, along with my comments appears below. (The Synertek notes are in bold face. My comments are normal type.)

1. The **Improved High Speed Cassette** read/write is significantly better than before. I was able to write and read quite constantly and was able to produce a tape on one type of recorder and read it on another. The volume/tone range was much wider. Whereas before you had to be right on for any chance of success, now you can have a reasonable variation in volume and tone and still get a good read. This is particularly important when you are using different recorders with different characteristics. The two recorders I tested with were a Superscope C-190 and a Pioneer Centrex. These fairly high quality recorders have **not** worked reliably with the old V1.0 monitor. A suggestion I had made to Synertek back in June 1978 was to make the leader time variable. While the 8 seconds they had built-in in V1.0 is acceptable when you are only occasionally storing a program, it was much to long if you intended to use the tape service to save small chunks of data - mailing list information for example. The above note says that the leader time is now maintained in ram and can be changed by the user if necessary. Since I did not have the listing or additional information, I was not able to test this out. But, assuming it does work, this can be a very significant improvement. some programs I have written require a lot of extra code simply to get around the "fixed" leader problem. They should be much simpler now, since I should be able to set the leader time in ram and then use the tape cassette routines directly.

2. **KIM read. Read routine improved.** This has been one of the biggest problems for the SYM-1 since it release. The V1.0 monitor had a simple, but powerful, bug. It made an invalid test for the KIM format "end-of-data" character, and treated the legal 32 46 ASCII pair as an ASCII "/", thereby terminating prematurely whenever it encountered a "2F" in the data. This made the KIM format mode of the SYM-1 essentially useless. This has been fixed in the new version. This means that it is now possible to distribute software, data bases, source files, etc. between the KIM-1, SYM-1 and AIM 65 using the common KIM format.

3. **Beeper frequency adjusted for maximum output.** I'll take their word for this. It does sound a little louder, but then I had never had any trouble with the beeper in V1.0

4. **During the VERIFY command a BREAK key will stop printout without printing an error message.** I didn't test this minor improvement, but it is nice to keep error messages for real errors.

5. **BREAK key is looked for on current loop interface.** If you are using a teletype device, it is handy to have the BREAK key work, so this change is definitely good.

6. **Log-on changed to SY1.1.** Yes.

7. **After paper tape load the error message count is displayed.** I do not have any paper tape facility to test this, but it is a minor improvement.

8. **Ability to return to a higher lever program (left arrow).** I do not quite understand what this is supposed to mean, but I am sure when additional documentation is available it will make sense.

9. **Cassette file I.D. displayed on left digit seven segments.** This is both cute and useful. They have simply taken the ID value and put it out on the leftmost digit. It does take a bit of deciphering though. The figure below shows the value of each segment on the display. These must be separately read and then added together to get the file ID. It is useful when you are searching the tape for a particular tape ID.

10. **Unwrite-protect routine added to cassette logic.** Again, I could not test this due to zero documentation.

11. **Register name improvement on display during R command.** Hooray! Now the display shows the register name, not a "hard-to-remember-and-interpret" arbitrary number to identify which register you are examining. P for program counter; S for stack; F for flags; A for A register, to represent an X for the X register; and Y for the Y register. A simple but very nice improvement.

12. **Debug-on will not cause ram to be write protected.** I did not test this, but it sounds reasonable.

That's the good news.

The Bad News

The bad news isn't all that bad, but should be considered. First, the changes to the Supermon do move some code around and change some "internal" entry points. Although the Synertek programmer I talked to said that this was not going to be very important since the main entry points were not touched, I found the first program I tried to run, the SYNC generator from the Reference Manual, would not work since two of the routines it requires have moved. How great a problem will this be? It is difficult to guess. I haven't seen the listings and do not know what routines were changed and also do not know how often other programmers have used them directly. It will be a problem for anyone who is trying to make program for distribution since there may be a requirement for two versions - one for V1.0 and another for V1.1 - and this adds to the expense and can cause distribution problems. Hopefully, the number of routines affected is small and isn't a big problem - but at present, "Who knows?".

Second, the V1.1 does use up some (most?, all?) of the Scratch Pad RAM in the System RAM. While this is not necessarily a big problem for future programs, it may cause problems for existing programs which use this previously available resource. Care will have to be taken when transferring programs from V1.0 to V1.1 to take this change in scratch pad availability into account.

Third, Synertek does not seem to have a policy yet for how the new V1.1 will be distributed. They are still waiting for feedback from myself and a couple of other users before committing to ROM, so it will be some time before any of the V1.1 are available at all. Then there is the question of systems already in the field or on dealer's shelves. Will there be a reasonable "exchange" policy, say Synertek's actual ROM production cost of $10-$15.00, or is some outlandish price going to be charged. I strongly feel that Synertek has the responsibility to offer the new V1.1 at the lowest price possible. Some of the changes they have made are not "cosmetic" or simple "improvements". They are basic "corrections" to their original "flawed" V1.0.

SYM-1 Codes

Ever wonder what the various codes were that the SYM used: keycode, ASCII code, and display code? You can look them up in the SYM manual in various places, but, why not let the SYM itself generate a display of these codes. The following program is an aid in establishing the relations between the three different codes. Start the program at 0000. The display goes blank, and when a key is depressed, the display will show key code, ASCII and display-scan code for a short time, and go blank again with a "beep".

Submitted by
Jan Skov
Majvaenget 7
DK-6000 Kolding
The Netherlands

```
                        SYM-1 CODE DISPLAY
                        JAN SKOV
                        FEBRUARY 1979

0000                            ORG   $0000

                        SYM SUBROUTINES

0000                    ACCESS  *     $8B86  SYSTEM RAM ACCESS
0000                    SPACE   *     $8342  OUTPUT SPACE TO DISPLAY
0000                    INCHR   *     $8A1B  INPUT CHARACTER
0000                    OUTCHR  *     $8A47  OUTPUT CHARACTER
0000                    OUTBYT  *     $82FA  OUTPUT BYTE
0000                    SCAND   *     $8906  SCAN DISPLAY
0000                    BEEP    *     $8972

0000 20 86 8B   START   JSR   ACCESS
0003 A2 06              LDXIM $06
0005 20 42 83   LOOP    JSR   SPACE
0008 CA                 DEX
0009 D0 FA              BNE   LOOP
000B 20 1B 8A           JSR   INCHR
000E 85 EF              STAZ  $00EF
0010 A9 2D              LDAIM $2D
0012 20 47 8A           JSR   OUTCHR
0015 A5 EF              LDAZ  $00EF
0017 20 FA 82           JSR   OUTBYT
001A AD 42 A6           LDA   $A642   DISPLAY BUFFER
001D 20 FA 82           JSR   OUTBYT
0020 A2 0B              LDXIM $0B
0022 86 EE              STXZ  $00EE
0024 86 ED              STXZ  $00ED
0026 20 06 89   LOOPA   JSR   SCAND   DISPLAY AND
0029 C6 ED              DECZ  $00ED   TIMER LOOP
002B D0 F9              BNE   LOOPA
002D C6 EE              DECZ  $00EE
002F D0 F5              BNE   LOOPA
0031 20 72 89           JSR   BEEP
0034 4C 00 00           JMP   START
```

A SIMPLE 24 HOUR CLOCK FOR THE AIM 65

Marvin L. De Jong
Department of Math-Physics
The School of the Ozarks
Point Lookout, MO 65726

The program whose listings are given in the AIM 65 disassembly format is a 24 hour clock that displays the time in hours, minutes, and seconds on the six right-most digits of the 20 character AIM 65 display. AIM 65 owners can load the program directly from the listings using the mini-assembler in the AIM 65 monitor. The program listings were taken directly from the thermal printer on the AIM 65.

The principal reason for writing the program was to experiment with the interval timers on the 6522 VIA. One advantage of the so-called T1 timer on the 6522 is that it can produce equally spaced interrupts, independent of the time necessary to complete an instruction and the time necessary to process the interrupt. SYM-1 owners may also use the program with only minor modifications, since the addresses of the various registers and counters in the 6522 chips are the same for these two computers. SYM-1 owners will have to change the display routines, however.

A brief description of the program follows. The first five instructions set up the interrupt vectors for the AIM 65. The next eight instructions set up the 6522 VIA for the T1 timer in the free running mode, enable the T1 interrupt, and set the time interval to $C34E = 49,998$_{10}$ clock cycles. This number, plus the two clock cycles necessary to restart the timer, represent 50,000 clock cycles or 0.05 seconds. Thus, the time between interrupts is exactly 50,000 clock cycles. Twenty interrupts give an interval of 10^6 clock cycles, or one second with a one MHz clock frequency. Location $0000 serves as register for the count-to-twenty interrupts process. It starts at $EC and advances to $00 before the seconds location is incremented.

The interrupt routine from $0300 to $033C is very similar to the clock program by Charles Parsons in THE FIRST BOOK OF KIM. The only difference is that the timers do not need to be restarted in the interrupt routine. Only the interrupt flag needs to be cleared before returning from interrupt. This is accomplished by the LDA A004 instruction at $0337.

The program from $0226 to $0254 is the display routine from the AIM 65. First the seconds, minutes, and hours located in $0001, $0002, and $0003 respectively, are relocated, then converted to ASCII, and finally output to the display by the JSR EF7B. Many kinds of hex to ASCII routines are possible here. I simply rotated nibble after nibble into the low order nibble of location $0004 and added $30 to convert to ASCII.

AIM 65 owners may be interested in the output routine. Of all the subroutines mentioned in the "User's Guide" the one I used is not mentioned directly. Basically it takes an ASCII character in the accumulator and outputs it to the display digit between $00 and $13 (20 character display) identified by the contents of the X register. It also requires a one in bit seven of the accumulator. Otherwise you get the cursor. So I did a ORA $80 with the ASCII character in the accumulator before jumping to the subroutine at $EF7B.

I checked the clock up against WWV and found it was off by about 0.024%, which is substantial if you wish to keep time over the long term. I decreased the $4E byte location $0216 to $42 and now it appears to be off by only 0.00063%. Of course, these timing errors, though small, tend to accumulate giving an error of about 0.5 seconds in 24 hours.

To start the timer, load the hours, minutes, and seconds locations with the time at which you intend to start, wait for this time, then start the program. Of course, there are much more meaningful applications to this program than simply displaying the time. One could record the time at which transistions on the I/O pins occur for example. Have fun.

```
0200 78    SEI
0201 A9    LDA  #00
0203 8D    STA  A404
0206 A9    LDA  #03
0208 8D    STA  A405
020B A9    LDA  #C0
020D 8D    STA  A00E
0210 A9    LDA  #40
0212 8D    STA  A00B
0215 A9    LDA  #4E
0217 8D    STA  A006
021A A9    LDA  #C3
021C 8D    STA  A005
021F A9    LDA  #EC
0221 85    STA  00
0223 58    CLI
0224 00    BRK
0225 EA    NOP
0226 A5    LDA  01
0228 85    STA  04
022A A5    LDA  02
022C 85    STA  05
022E A5    LDA  03
0230 85    STA  06
0232 A2    LDX  #13
0234 8A    TXA
0235 48    PHA
0236 A0    LDY  #04
0238 A5    LDA  04
023A 29    AND  #0F
023C 18    CLC
023D 69    ADC  #30
023F 09    ORA  #80
0241 20    JSR  EF7B
0244 46    LSR  06
0246 66    ROR  05
0248 66    ROR  04
024A 88    DEY
024B D0    BNE  0244
024D 68    PLA
024E AA    TAX
024F CA    DEX
0250 E0    CPX  #0E
0252 B0    BCS  0234
0254 4C    JMP  0226
```

```
0300 48    PHA
0301 E6    INC  00
0303 D0    BNE  0337
0305 F8    SED
0306 18    CLC
0307 A5    LDA  01
0309 69    ADC  #01
030B 85    STA  01
030D C9    CMP  #60
030F 90    BCC  0333
0311 A9    LDA  #00
0313 85    STA  01
0315 18    CLC
0316 A5    LDA  02
0318 69    ADC  #01
031A 85    STA  02
031C C9    CMP  #60
031E 90    BCC  0333
0320 A9    LDA  #00
0322 85    STA  02
0324 18    CLC
0325 A5    LDA  03
0327 69    ADC  #01
0329 85    STA  03
032B C9    CMP  #24
032D 90    BCC  0333
032F A9    LDA  #00
0331 85    STA  03
0333 A9    LDA  #EC
0335 85    STA  00
0337 AD    LDA  A004
033A D8    CLD
033B 68    PLA
033C 40    RTI
```

24 HOUR AIM CLOCK

BY MARVIN L. DE JONG
FEBRUARY 1979

```
0200                    ORG   $0200

0200 78         START   SEI           SET INTERRUPT DISABLE
0201 A9 00              LDAIM $00     SETUP INTERRUPT VECTORS
0203 8D 04 A4           STA   $A404   FOR 6522
0206 A9 03              LDAIM $03     POINT TO ADDRESS 0300
0208 8D 05 A4           STA   $A405
020B A9 C0              LDAIM $C0     SETUP VIA 6522 FOR TIMER 1
020D 8D 0E A0           STA   $A00E   IN FREE RUNNING MODE
0210 A9 40              LDAIM $40
0212 8D 0B A0           STA   $A00B
0215 A9 4E              LDAIM $4E     SET LOW BYTE OF TIMER
0217 8D 06 A0           STA   $A006
021A A9 C3              LDAIM $C3     SET HIGH BYTE OF TIMER
021C 8D 05 A0           STA   $A005
021F A9 EC              LDAIM $EC     SET 20 INTERRUPT COUNTER
0221 85 00              STA   $0000   IN LOCATION 0000
0223 58                 CLI           ENABLE INTERRUPTS
0224 00                 BRK           RETURN TO MONITOR
0225 EA                 NOP

0226 A5 01      DISPLY  LDA   $0001   MOVE DIGITS TO BE DISPLAYED
0228 85 04              STA   $0004   FOR SAFE KEEPING
022A A5 02              LDA   $0002
022C 85 05              STA   $0005
022E A5 03              LDA   $0003
0230 85 06              STA   $0006
0232 A2 13              LDXIM $13     LOAD DISPLAY POSITION POINTER
0234 8A         LOOP    TXA           PUT X VALUE INTO A
0235 48                 PHA           SAVE ON STACK
0236 A0 04              LDYIM $04     SET TO SHIFT FOUR POSITIONS
0238 A5 04              LDA   $0004   GET LEAST SIGN DIGIT REMAINING
023A 29 0F              ANDIM $0F     MASK TO SINGLE CHARACTER
023C 18                 CLC           CLEAR
023D 69 30              ADCIM $30     CONVERT 0-9 TO ASCII 0 - 9
023F 09 80              ORAIM $80     BIT 80 MUST BE ON FOR AIM
0241 20 7B EF           JSR   $EF7B   AIM OUTPUT ROUTINE
0244 46 06      SHIFT   LSR   $0006   SHIFT TO GET HIGH HALF OF
0246 66 05              ROR   $0005   DIGIT INTO POSITION
0248 66 04              ROR   $0004
024A 88                 DEY           DECREMENT FOUR SHIFT COUNTER
024B D0 F7              BNE   SHIFT   KEEP ON SHIFTING
024D 68                 PLA           RESTORE X FROM STACK
024E AA                 TAX
024F CA                 DEX           DECREMENT POSITION POINTER
0250 E0 0E              CPXIM $0E     TEST 6 DIGITS OUTPUT
0252 B0 E0              BCS   LOOP    MORE TO DO
0254 4C 26 02           JMP   DISPLY  DONE. NOW START OVER AGAIN.
```

24 HOUR CLOCK INTERRUPT SERVICE

```
0300                    CRG    $C3CC

0300 48          INTRPT PHA            COME HERE ON TIMER INTERRUPT
0301 E6 0C              INC    $0CC0  SAVE A REG AND BUMP COUNTER IN 000C
0303 D0 32              BNE    IDONE  DONE WITH INTERRUPT
0305 38                 SEC           SET DECIMAL MODE FOR CALCULATIONS
0306 18                 CLC
0307 A5 C1              LDA    $0CC1  BUMP ONE SECOND COUNTER
0309 69 01              ADCIM  $01    BY ADDING 1 WITH CARRY
030B 85 01              STA    $CC01  SAVE
030D C9 60              CMPIM  $60    TEST SIXTY SECONDS
030F 90 22              BCC    NOTMIN NOT A MINUTE
0311 A9 00              LDAIM  $00    A MINUTE
0313 85 01              STA    $0001  ZERO SECOND COUNTER
0315 18                 CLC           THEN BUMP MINUTES
0316 A5 02              LDA    $CC02  GET MINUTES COUNTER
0318 69 01              ADCIM  $01    AND BUMP
031A 85 02              STA    $0002  SAVE
031C C9 60              CMPIM  $60    TEST HOUR
031E 90 13              BCC    NOTMIN NOT AN HOUR YET.
0320 A9 00              LDAIM  $C0    AN HOUR, SO ZERO MINUTES
0322 85 02              STA    $00C2
0324 18                 CLC           THEN FIX HOURS
0325 A5 03              LDA    $00C3
0327 69 01              ADCIM  $C1
0329 85 03              STA    $0003
032B C9 24              CMPIM  $24    TEST 24 HOURS
032D 90 04              BCC    NOTMIN NOT 24 HOURS
032F A9 0C              LDAIM  $0C    AT 24 HOURS RESET TO ZERO
0331 85 03              STA    $0003

0333 A9 EC       NOTMIN LDAIM  $EC    RESET 20 INTERRUPT COUNTER
0335 85 00              STA    $000C

0337 AD 04 A0    IDONE  LDA    $A004  RESTART TIMER BY READING
033A D8                 CLD           CLEAR DECIMAL MODE
033B 68                 PLA           RESTORE A REGISTER
033C 40                 RTI           RETURN FROM INTERRUPT
```

AN AIM 65 USER's NOTES

Joe Burnett
16492 F. Tennessee Avenue
Aurora, CO 80012

The AIM 65 Microcomputer, made by Rockwell, is one of the newest, most versatile home computers available today. At the time of this writing (January 1979), it sells for $375. For this you get the complete computer, with a 20 character alphanumeric display, full size alphanumeric keyboard, a printer which uses inexpensive calculator type paper, 1K of RAM and 8K ROM-resident programming. Options include the ability to add 3K more memory, a 4K assembler, and an 8K Basic interpreter, all on-board, simply by purchasing them and plugging them in. An "application" connector and an "expansion" connector accept standard 44 pin edge connectors, and allow the control and I/O of two cassette units and a teletype, as well as off-board additional memory. On-board programming (ROM-resident) gives you the ability to display memory in either hex or mnemonic, alter memory, edit programming, turn the printer on and off, display registers, and enter any of the many resident subroutines. With cassette units connected, you can read or write to either one, and set up the AIM 65 to handle KIM-1 format (X1 or X3) or the AIM 65 format software. The AIM 65 will file and search cassette tapes, and the front panel alphanumeric display lets you know the status of the operation in progress as well as the block of data being read or written. Three keys on the keyboard (F1, F2, and F3) enable user defined functions through programmed jump instructions, and are a nice feature. Physically, the computer circuit board itself is ten inches deep by twelve inches wide, and the keyboard (which attaches through a supplied ribbon cable) is four inches deep by twelve inches wide. Included with the computer is a roll of paper for the printer, "feet" for the computer circuit board and the keyboard circuit, a User's Guide manual, an R6500 Programming manual, a System Hardware manual, a Programming Reference Card, an AIM 65 Summary Card, and a large schematic diagram, as well as the warranty card (don't forget to mail this in).

Software Compatibility

As with any new product, there are some problems. One is with the KIM-1 software. The KIM-1 is a very basic computer, and the AIM 65 is sophisticated by comparison. An example of the problem with the software is the KIM-1 "PLFASE" program. "PLFASE" loads data into memory locations which either are dedicated for use by the AIM 65, or are not present in the AIM 65. Consequently, although the AIM 65 can be initialized to accept KIM-1 programming, check the listing before you try to do it. It'll save you a lot of time and frustration. The AIM 65 User's Guide Manual includes a detailed memory map which you can use to determine (from a program listing) whether or not the program you're trying to load will in fact load as advertised.

Some Cassette Control Problems

A second problem is with the cassette unit control circuitry. There are actually two circuits in the AIM 65 for each cassette unit, and although Rockwell made an attempt to cover all eventualities, they didn't succeed. The first circuit makes use of an integrated circuit relay driver, which puts a low (ground) at the cassette control output pin of the "application" connector when the computer toggles the cassette unit "on". The second circuit is a transistor switch which is biased on when the computer toggles the cassette unit "on". The problem arises in that not all cassette units use a positive supply voltage with the negative line common (connected to the cassette unit frame). General Electric, for example, typically connects the positive side of the battery (or AC adapter) to the cassette unit frame, and uses negative voltage for the motor and electronic circuitry. At first glance, this doesn't look like a problem; after all, you only need to supply a closure to the remote switch line, and the cassette unit will run, right? Well, not quite. If you connect your GE cassette unit to the relay driver output pin, and the computer control has the cassette unit toggled "off", the cassette unit won't shut off. This is because you've put a negative voltage (from the cassette unit) at a point which has a nearly equal positive voltage (from the AIM 65), and the result is close enough to zero volts that the cassette unit motor runs even though the computer indicated that an "off" condition exists. Okay, so what about the transistor switch? Figure 9-4 of the User's Guide manual shows how to connect the wires. And the cassette unit won't run. At this point you're most likely very annoyed and confused (I know I was). The reason that the computer won't control the cassette unit is that (1) figure 9-4 of the User's Guide Manual is in error; the positive voltage from the cassette unit battery should go to pin "F", and the motor line should go to pin "F", of the "application" connector; and (2) the transistor does not have the voltages necessary to make it work, even after the wires are properly connected. If you look at the schematic diagram, you'll see that the transistor switch in the computer gets its operating voltage from the circuit it's controlling. To make it work, the transistor must have the proper bias (voltage between base and omitter), and to get this a common ground must exist between the computer power supply and the cassette unit power supply. It would seem that all that would be necessary would be to connect the emitter of the transistor (pin "F" of the "application" connector) to ground. Now the cassette unit will run and stop in response to computer control---until you plug in the ear and/or mic lines. When you do this, and the transistor turns on, you create a short circuit across the battery (or AC adapter) of the cassette unit. The reason is that when you wired up the ear/mic lines, you connected one side to ground on the 44 pin edge connector, and now the current finds a path through the cassette electronic circuitry, and everything stops. Under normal conditions, the remote switch on the cassette unit microphone is isolated from everything, so no problem exists. When you make the return line to the remote switch and the ear/mic line return common, a short circuit occurs. So what do you do now? Simulate an isolated switch, similar to what the microphone has. A relay is the only way, if you're going to control the cassette unit with the computer. Since my AIM 65 is still in the warranty period, I have not modified it as I'd like to. However, once the warranty period expires, I'm going to install two relays on the circuit board and use the transistor switches to control them. Then it won't matter what kind of motor control the cassette unit uses; I'll have the isolated switch action required to control any cassette unit, regardless of the polarity of the voltages involved.

At the time of this writing, neither the Assembler nor the BASIC interpreter is available from my distributor. This means that any programming I do has to be done using mnemonic codes. Although the documentation in the User's Guide is very good, the sample programs shown appear to have been produced with the use of an Assembler. An example is on pages 7-82 and 7-83. This program is intended to display and print an assembled message, but the information on how to prepare the message for storage in memory is absent. So, if you input this program you'll be " all dressed up with nowhere to go". The program shown below will allow you to input a message, and then retrieve it, all with the "bare bones" (1K RAM) AIM 65. How you use this is up to you. It could be just "for show", or you can modify it as desired and include it in more complex routines involving user interaction with the computer. This program does feature single key access (user function key F1, F2, or F3). Key F1 allows you to write to memory; key F2 retrieves the entire message; and key F3 retrieves the message a line at a time, with the space bar being used to advance the display to the next line of the message. The maximum length of the message is 13½ lines. An asterisk is typed at the end of the message when it is written to memory, which takes the computer out of the loop in all of the modes.

I hope the information in this article helps you avoid some of the problems and frustrations I've experienced. Enjoy your AIM 65. I'm having a lot of fun with mine, and I'm still learning what it's capabilities are.

```
                        WRITE TO MEMORY PROGRAM
                        JOE BURNETT
                        WITH MODS BY MIKE ROWE
                        APRIL 1979

0000                            ORG   $0000

                        AIM SUBROUTINES

0000                    CRCK    *     $EA24  DUMP PRINT BUFFER
0000                    CRLF    *     $E9F0  CARRIAGE RETURN/LINE FEED
0000                    INALL   *     $E993  INPUT FROM ANY DEVICE
0000                    OUTALL  *     $E9BC  OUTPUT TO ANY DEVICE

                        ASCII CHARACTER

0000                    SPACE   *     $0020  SPACE CHARACTER
0000                    ASTER   *     $002A  ASTERISK CHARACTER

                        WRITE MESSAGE TO MEMORY

0000 20 F0 E9  WRITE    JSR   CRLF    CLEAR DISPLAY
0003 A0 00             LDYIM  $00     INIT MEMORY POINTER
0005 A2 13     LINE    LDXIM  $13     INIT CHARACTER COUNTER
0007 20 93 E9  INPUT    JSR   INALL   GET AN INPUT CHARACTER
000A 99 00 02          STAY   $0200   STORE IN BUFFER
000D C9 2A             CMPIM  ASTER   TEST TERMINATOR
000F F0 47             BEQ    EXIT    IF YES, THEN DONE
0011 C8                INY            BUMP POINTER
0012 CA                DEX            DECR CHARACTER COUNTER
0013 D0 F2             BNE    INPUT   IF NOT ZERO, GET MORE
0015 20 24 EA          JSR    CRCK    LINE FULL, SO PRINT IT
0018 4C 05 00          JMP    LINE    GET NEXT LINE

                        READ ENTIRE MESSAGE

001B 20 F0 E9  REM      JSR   CRLF    CLEAR DISPLAY
001E A0 00             LDYIM  $00     INIT MEMORY POINTER
0020 A2 13     RLINE   LDXIM  $13     INIT CHARACTER COUNTER
0022 B9 00 02  RCHAR   LDAY   $0200   GET CHARACTER FROM MEMORY
0025 C9 2A             CMPIM  ASTER   TEST FOR TERMINATOR
0027 F0 2F             BEQ    EXIT    IF YES, THEN DONE
0029 20 BC E9          JSR    OUTALL  ELSE, DISPLAY CHARACTER
002C C8                INY            BUMP MEMORY POINTER
002D CA                DEX            DECR. CHARACTER COUNTER
002E D0 F2             BNE    RCHAR   IF NOT ZERO, GET NEXT CHARACTER
0030 20 24 EA          JSR    CRCK    ELSE, PRINT LINE
0033 4C 20 00          JMP    RLINE   THEN CONTINUE
```

```
                    READ MESSAGE ONE LINE AT A TIME

0036 20 F0 E9  ONELIN JSR   CRLF    CLEAR DISPLAY
0039 A0 00             LDYIM $00    INIT MEMORY POINTER
003B A2 13     OLINE   LDXIM $13    INIT CHARACTER COUNTER
003D B9 00 02  OCHAR   LDAY  $0200  GET CHARACTER FROM MEMORY
0040 C9 2A             CMPIM ASTER  TEST TERMINATOR
0042 F0 14             BEQ   EXIT   IF YES, THEN DONE
0044 20 BC E9          JSR   OUTALL ELSE, PRINT CHARACTER
0047 C8                INY          BUMP MEMORY POINTER
0048 CA                DEX          DECR CHARACTER COUNTER
0049 D0 F2             BNE   OCHAR  IF NOT ZERO, CONTINUE
004B 20 93 E9  WAIT    JSR   INALL  ELSE WAIT FOR A SPACE
004E C9 20             CMPIM SPACE  FROM KAYBOARD TO CONTINUE
0050 D0 F9             BNE   WAIT   NOT A SPACE
0052 20 24 EA          JSR   CRCK   SPACE, SO PRINT
0055 4C 3B 00          JMP   OLINE  THEN GET NEXT LINE

                    COMMON EXIT ROUTINE TO CLEAN UP
                    THE DISPLAY AND RETURN TO MONITOR

0058 20 F0 E9  EXIT    JSR   CRLF   OUTPUT TO BLANK LINES
005B 20 F0 E9          JSR   CRLF
005E 00                BRK          THEN EXIT TO MONITOR

                    USER FUNCTION DEFINITIONS

010C                   ORG   $010C

010C 4C 00 00          JMP   WRITE  F1 TO WRITE MESSAGE
010F 4C 1B 00          JMP   REM    F2 TO READ ENTIRE MESSAGE
0112 4C 36 00          JMP   ONELIN F3 TO READ ONE LINE AT A TIME
```

A DIGITAL CLOCK PROGRAM FOR THE SYM-1

Chris Sullivan
9 Galsworthy Place
Bucklands Beach
Aukland, New Zealand

The SYM-1 is a one board hobbyist computer similiar to the KIM but with a number of additional features. Since buying the SYM-1 I have had a great deal of fun playing around with both the software and hardware sides of it. The SYM-1 monitor, Supermon, is an incredible monitor in 4K ROM, some of it's subroutines are called by the following program.

This program started off as a lesson in familiarity with the 6502 instruction set and using the Supermon subroutines to advantage, but the present version has been modified many times in order to increase the clock accuracy and, as my knowledge of the 6502 instruction set grows, increase coding efficiency. To use it one should start execution at :200. Then enter an "A" or "P" (Shift ASCII 5 0) to signify AM or PM. Then enter the hours (two digits), the program then outputs a space to separate the hours from the minutes. Finally enter 2 digits to signify the minutes, the program will then increment the minutes by 1, and begin the clock sequence. This slight quirk makes it easier to set the clock using another clock, set up the "A" or "P", hours and first digit of the minutes, then enter the last digit of the minutes as the seconds counter of your setting clock reaches 0.

There is another slight quirk in that the clock counts "A11 59", "A12 00", "A12 01", ..., "A12 59", "P01 00","P01 01" ..... This simplifies the programming and means that 12:30 near midday is in fact, 12:30 AM according to this clock! However this is not likely to confuse many people.

After setting up the initial time, the program adds 1 to the minutes and then carries on any carry into the hours, possibly changing "A" to "P" or vice versa. This section of the program could be made more efficient with full exploitation of the 6502 instruction set. The last section in the program is a 1 minute delay. I have rewritten this section many times in a search for an accurate 1 minute delay. The first part is a double loop which also scans the clock display, this loop takes about 59.8 seconds. The second part is a double loop to "tweak" the delay up to 60 seconds and consists of 2 delays using the onboard 6532 timer. This timer is initialised in 1 of 4 memory locations, specifying ÷1024, ÷64, ÷8, or ÷1 timing, e.g., the location to write to if one wants ÷1024 timing is A417. This location thus initialised is counted down in the 6532. The program reads this value until it becomes negative, at which time the delay is over.

Some improvements to the program could be made, for example better coding in the increment minutes section. One could also add an alarm feature, possibly using the on board beeper. The The section to update the time by one minute could be used as a part of a background real time clock, being called by a once-a-minute hardware interupt generated by an on board 6522 timer chip. Once a minute, processing would be interupted for 100 cycles or so in order to update the real time clock. Such clocks have many uses, one of which is to ensure that certain number-crunching programs don't get tied down in big loops.

This improved version occupies less RAM by using jumps to INBYTE rather than INCHAR and messy bit manipulations. The delay routine has been improved to use the on board 6532 timer, and also give greater resolution and hence greater timing accuracy.

Editor's Note: This program is present primarily for its value in showing how to access the SYM's monitor for some of the routines. It is not an "optimal" program for a 24 hour clock, but should be a good starting point for owners of SYMs who wish to write similar programs.

```
                        SYM-1 ELECTRONIC CLOCK

                        BY CHRIS SULLIVAN  AUGUST 27, 1978

                               ORG    $0200

                        SPACE  *      $0020  ASCII SPACE
                        ACCESS *      $8B86
                        INCHAR *      $8A1B
                        INBYTE *      $81D9
                        OUTCHR *      $8A47
                        OUTBYT *      $82FA

0200 20 86 8B   BEGIN   JSR    ACCESS
0203 20 1B 8A           JSR    INCHAR GET A OR P
0206 85 00              STAZ   $00
0208 18                 CLC
0209 20 D9 81           JSR    INBYTE GET HOURS
020C 85 01              STAZ   $01
020E A9 20              LDAIM  SPACE  SPACE CHARACTER
0210 20 47 8A           JSR    OUTCHR OUTPUT A SPACE
0213 20 D9 81           JSR    INBYTE GET MINUTES
0216 85 02              STAZ   $02
0218 F8                 SED            SET DECIMAL MODE FOR REMAINDER OF PROGRAM
```

```
                HAVING SET THE INITIAL TIME (LESS 1 MINUTE)
                UPDATE THE TIME:

0219 18         TIMLOP CLC
021A A5 02             LDAZ  $02    GET MINUTES
021C 69 01             ADCIM $01    INCREMENT
021E 85 02             STAZ  $02
0220 38                SEC
0221 E9 60             SBCIM $60    TEST IF NEW HOUR
0223 F0 03             BEQ   TIMEX
0225 4C 50 02          JMP   NORSET IF NOT A NEW HOUR

0228 A9 00      TIMEX  LDAIM $00
022A 85 02             STAZ  $02    SET MINUTES TO 00
022C 18                CLC
022D A5 01             LDAZ  $01
022F 69 01             ADCIM $01    INCR HOURS
0231 85 01             STAZ  $01
0233 38                SEC
0234 E9 13             SBCIM $13    TEST HOURS = 13
0236 F0 03             BEQ   TIMEY
0238 4C 50 02          JMP   NORSET

023B A9 01      TIMEY  LDAIM $01    YES, SET HOURS TO 1
023D 85 01             STAZ  $01
023F A5 00             LDAZ  $00    GET A OR P
0241 49 50             EORIM $50    ASCII P
0243 F0 07             BEQ   TIMEZ  IS 00 = ASCII P?
0245 A9 50             LDAIM $50    NO, THEN SET 00 TO P
0247 85 00             STAZ  $00
0249 4C 50 02          JMP   NORSET
024C A9 41      TIMEZ  LDAIM $41    YES, THEN SET 00 TO A
024E 85 00             STAZ  $00

0250 A5 00      NORSET LDAZ  $00    GET A OR P
0252 20 47 8A          JSR   OUTCHR
0255 A5 01             LDAZ  $01    GET HOURS
0257 20 FA 82          JSR   OUTBYT
025A A9 20             LDAIM SPACE
025C 20 47 8A          JSR   OUTCHR
025F A5 02             LDAZ  $02    GET MINUTES
0261 20 FA 82          JSR   OUTBYT
0264 D8                CLD          CLEAR DECIMAL MODE
0265 A2 C0             LDXIM $C0    SETUP FOR ALMOST 60 SEC WAIT
0267 A0 7D      WAITA  LDYIM $7D    COUNTER
0269 A9 01      WAITB  LDAIM $01    NON-DISPLAYING CHARACTER
026B 20 47 8A          JSR   OUTCHR REFRESH DISPLAY
026E 88                DEY
026F D0 F8             BNE   WAITB  LOW ORDER COUNTER
0271 CA                DEX          HIGH ORDER COUNTER
0272 D0 F3             BNE   WAITA
0274 A2 02             LDXIM $02    TWEAK TIME UP TO 60 SECONDS
0276 A9 4D      WAITC  LDAIM $4D
0278 8D 17 A4          STA   $A417  DIVIDE BY 1024 TIMER
027B AD 06 A4  WAITD   LDA   $A406  REGISTER OF 6532
027E 10 FB             BPL   WAITD
0280 CA                DEX
0281 D0 F3             BNE   WAITC
0283 F8                SED
0284 4C 19 02          JMP   TIMLOP
                VERIFY from 0200 thru 0286 is 356F.
```

The following subroutines called form part of the SYM-1's SUPERMON monitor:

ACCESS Enables the user program to write to system RAM, i.e. the RAM contained on the 6532. It is necessary to call ACCESS before calling most of the other system subroutines.

INCHAR Get one ASCII charcter from the input device (here the hex keypad) and return with it in the A register.

INBYTE Get two ASCII characters from the input device, using INCHAR and pack into a single byte in the A register.

OUTCHR Output the ASCII data in the A register to the output device (here the six digit LED display).

OUTBYT Convert the byte in the A register into two ASCII characters and output these to the output device.

Location A417 is used to initialise the 6532 timer to count down from the value stored in A417, with a divide by 1024 cycles. Thus the timer register on the 6532 is decremented by one every 1024 clock cycles. The timer register sits at location A406, and the time is considered to be "up" when the value at A406 becomes negative.

SUPER HI-LO FOR THE SYM-1

Jack Gieryic
2041 138th Ave. N.W.
Andover, MN 55303

Super HI-Lo has a new twist to the game. This program fits into the standard 1K SYM and execution begins at location 200. The left two LED digits are your upper limit (initialized to 99) and the middle two digits are your lower limit (initialized to 00). SYM picks a random number and you attempt to guess it. Your attempt count is seen in the right two digits. The right digit will blink when it's your last guess.

After entering the command GO 200 CR press any key to start the contest. Enter your two digit guess (decimal only) and hit the "A" key. Win or loose you get an appropriate message at the end after which the LED's go blank. Hit any key and you are ready for a second game. If you didn't guess the number then you will be given one more chance in the next game. If you are lucky enough to guess the number then you will have one less chance the next game.

For you SYMMERS who are interested in taking things one step further, you will find MESSAG an interesting subroutine you may want to incorporate in your own programs. This code is entirely relocatable except for the first four instructions which must be calculated if the code is moved. The routine uses page zero locations OD, OE, OF and 10, but you can change that too if necessary. The A and X registers contain the message buffer address per comments in the program. This message buffer contains segment codes which will light up any combination of LED segments.

Refer to Figure 4-6 Keyboard/Display Schematic in your reference manual for the LED segments in the lower right corner. Segment "a" is turned on by setting bit 0 to a one in a message buffer entry. Segment "b" is controlled by bit 1 and so on with segments c, d, e, f, g and the decimal point. Thus a hex 5C is a lower case O (segments c, d, e, and g). Feel free to change either message but don't forget to add a few OO characters at the start and end of your message. If you relocate the message buffer then change the register parameters prior to the call to MESSAG.

One other note on the program. By changing the value at location 206 you can alter the rate at which the right LED will blink when you reach your last chance.

```
                        SYM SUPER HI-LO
                        JOHN GIERYIC
                        APRIL 1979

                        SYM REFERENCES

035E                    KYSTAT *    $896A
035E                    ACCESS *    $8B86
035E                    OUTBYT *    $82FA
035E                    SCAND  *    $8906
035E                    KEYQ   *    $8923
035E                    GETKEY *    $88AF
035E                    ASCNIB *    $8275
035E                    DISBUF *    $A640
035E                    RDIG   *    $A645

                        MESSAGE POINTERS

035E                    MFAIL  *    $0360
035E                    MSUCC  *    $0380

0000                          ORG   $0000

0000 00                 UPP    =    $00    UPPER NUMBER
0001 00                 LOW    =    $00    LOWER NUMBER
0002 00                 ACNT   =    $00    ATTEMPT COUNT
0003 00                 RAN    =    $00    RANDOM NUMBER 2 - 98
0004 00                 TEMP   =    $00
```

```
0005 00          UGES    =       $00     GUESS UNITS
0006 00          TGES    =       $00     GUESS TENS
0007 00          BLINK   =       $00     BLINK FLAG 1 = BLINK
0008 00          TDIG    =       $00     SAVE RDIG
0009 00          DARK    =       $00     1 = DARK
000A 00          LATT    =       $00     ATTEMPT LIMIT
000B 00          ONOFF   =       $00     BLINKING
000C 00          BLIM    =       $00     BLINKING LOOP COUNT INIT.
000D 00          COUNT   =       $00
000E 00          LOOPA   =       $00
000F 00          LOOPB   =       $00
0010 00          CLIM    =       $00     MESSAGE LIMIT

0200                     ORG     $0200   PROGRAM ORIGIN

0200 20 86 8B    BEGIN   JSR     ACCESS
0203 A9 60               LDAIM   $60     INIT BLINKING LOOP LIMIT
0205 85 0C               STA     BLIM
0207 A9 06               LDAIM   $06     INIT ATTEMPT COUNTER
0209 85 0A               STA     LATT

020B A9 63       TILL    LDAIM   $63     INIT UPPER LIMIT
020D 85 00               STA     UPP
020F A9 00               LDAIM   $00     INIT BLINK FLAG
0211 85 07               STA     BLINK
0213 85 01               STA     LOW     LOWER LIMIT
0215 85 02               STA     ACNT    ATTEMPT COUNT
0217 A9 01               LDAIM   $01
0219 85 03               STA     RAN     RANDOM NUMBER

021B E6 03       INCRAN  INC     RAN     INCREMENT RANDOM NUMBER
021D A5 03               LDA     RAN
021F C9 63               CMPIM   $63     IF EQUAL 99 DECIMAL
0221 D0 04               BNE     KEYIN
0223 A9 02               LDAIM   $02     THEN RESET TO 2
0225 85 03               STA     RAN

0227 20 6A 89    KEYIN   JSR     KYSTAT  IS A KEY DOWN?
022A 90 EF               BCC     INCRAN  LOOP UNTIL ONE IS DOWN
022C A5 00       LIMITS  LDA     UPP     PUT UPPER, LOWER AND
022E 20 00 03            JSR     HTDEC   ATTEMPT COUNT IN
0231 20 FA 82            JSR     OUTBYT  DISPLAY BUFFER
0234 A5 01               LDA     LOW
0236 20 00 03            JSR     HTDEC
0239 20 FA 82            JSR     OUTBYT
023C A5 02               LDA     ACNT
023E 20 00 03            JSR     HTDEC
0241 20 FA 82            JSR     OUTBYT

0244 20 06 89    DISP    JSR     SCAND   LIGHT LED
0247 20 23 89            JSR     KEYQ    IF KEY IS DOWN,
024A D0 30               BNE     READK
024C A5 07               LDA     BLINK   IF BLINKING IS REQUESTED
024E C9 01               CMPIM   $01
0250 D0 F2               BNE     DISP
0252 A5 0B               LDA     ONOFF   IF TIME TO TURN CHARACTER ON
```

```
0254 D0 21            BNE   INCLOP
0256 A5 09            LDA   DARK    IF TURN CHAR. OFF
0258 C9 01            CMPIM $01
025A D0 0E            BNE   RIGHT
025C AD 45 A6         LDA   RDIG    THEN GET CHARACTER
025F 85 08            STA   TDIG    SAVE IT
0261 A9 00            LDAIM $00     SET RIGHT DIGIT BLANK
0263 8D 45 A6         STA   RDIG
0266 C6 09            DEC   DARK    SWITCH FLAG
0268 F0 07            BEQ   LCOUNT
026A A5 08      RIGHT LDA   TDIG    ELSE RESTORE RIGHT DIGIT
026C 8D 45 A6         STA   RDIG
026F E6 09            INC   DARK    SWITCH FLAG
0271 A5 0C      LCOUNT LDA  BLIM    RESET LOOP COUNTER
0273 85 0B            STA   ONOFF
0275 D0 CD            BNE   DISP
0277 E6 0B      INCLOP INC  ONOFF   INCR. LOOP COUNTER
0279 4C 44 02         JMP   DISP    LOOP

027C 20 AF 88   READK JSR   GETKEY  GET DEPRESSED KEY
027F 20 75 82         JSR   ASCNIB
0282 C9 0A            CMPIM $0A     IS IT "A" (ATTEMPT)
0284 F0 0B            BEQ   SETLOP  YES
0286 AA               TAX           NO
0287 A5 05            LDA   UGES    MOVE PREVIOUS KEY
0289 85 06            STA   TGES    TO TENS DIGIT
028B 8A               TXA
028C 85 05            STA   UGES    PUT NEW KEY INTO UNITS
028E 4C 44 02         JMP   DISP    LOOP

0291 A6 06      SETLOP LDX  TGES    SET LOOP INDEX (TENS)
0293 A9 00            LDAIM $00     INIT A REGISTER
0295 18               CLC           CLEAR CARRY FALG
0296 CA         DECX  DEX           DECR. X REG.
0297 30 04            BMI   ADUNIT  IF NEG, THEN FINISHED

0299 69 0A            ADCIM $0A     ELSE ADD 10
029B D0 F9            BNE   DECX     LOOP
029D 65 05      ADUNIT ADC  UGES    ADD UNITS VALUE
029F C5 03            CMP   RAN     COMPARE TO RANDOM
02A1 D0 03            BNE   ADUP
02A3 4C E4 02         JMP   SUCEED  GUESS = RANDOM

02A6 90 09      ADUP  BCC   TLOW
02A8 C5 00            CMP   UPP
02AA B0 0B            BCS   INCA
02AC 85 00      RUP   STA   UPP     REPLACE UPPER WITH GUESS
02AE 4C B7 02         JMP   INCA
02B1 C5 01      TLOW  CMP   LOW
02B3 90 02            BCC   INCA
02B5 85 01            STA   LOW     REPLACE LOWER WITH GUESS
02B7 E6 02      INCA  INC   ACNT    INCR. ATTEMPT COUNT
02B9 A5 02            LDA   ACNT    LIMIT REACHED?
02BB C5 0A            CMP   LATT
02BD D0 03            BNE   TEST    NO
02BF 4C D8 02         JMP   FAIL    YES = FAILURE
```

```
02C2 38          TEST   SEC
02C3 A5 0A              LDA    LATT    LAST ATTEMPT COMING UP
02C5 E5 02              SBC    ACNT
02C7 C9 01              CMPIM  $01
02C9 D0 0A              BNE    WAIT    NO
02CB E6 07              INC    BLINK   YES - INIT FOR BLINKING
02CD A5 0C              LDA    BLIM
02CF 85 0B              STA    ONOFF
02D1 A9 01              LDAIM  $01
02D3 85 09              STA    DARK
02D5 4C 2C 02   WAIT    JMP    LIMITS GO WAIT FOR NEXT ATTEMPT

02D8 E6 0A      FAIL    INC    LATT    FAILURE = INCR ATTEMPT LIMIT
02DA A2 03              LDXIM  MFAIL   / MESSAGE HI BYTE
02DC A9 60              LDAIM  MFAIL     MESSAGE LO BYTE
02DE 20 17 03           JSR    MESSAG  DISPLAY FAILURE MESSAGE
02E1 4C 0B C2           JMP    TILL    RESTART HI-LO

02E4 C6 0A      SUCEED  DEC    LATT    SUCCESS = DECR ATTEMPT LIMIT
02E6 A2 03              LDXIM  MSUCC   / MESSAGE HI BYTE
02E8 A9 80              LDAIM  MSUCC    MESSAGE LO BYTE
02EA 20 17 03           JSR    MESSAG  DISPLAY SUCCESS MESSAGE
02ED 4C 0B 02           JMP    TILL    RESTART HI-LO

                SUBROUTINE HTDEC

                ENTRY  JSR HTDEC
                THIS ROUTINE WILL CONVERT A HEX NUMBER
                TO DECIMAL.  UPON ENTRY THE A REGISTER CONTAINS
                THE NUMBER TO CONVERT.  UPON EXIT THE A REG.
                CONTAINS THE UNITS DIGIT AND THE X REGISTER
                CONTAINS THE TENS DIGIT.

0300                    ORG    $0300

0300 A2 00      HTDEC   LDXIM  $00     INIT TENS COUNT
0302 38                 SEC
0303 E9 0A      HTA     SBCIM  $0A     SUBTRACT 10 DECIMAL
0305 30 03              BMI    HTB
0307 E8                 INX            INCR. TENS DIGIT
0308 D0 F9              BNE    HTA
030A 69 0A      HTB     ADCIM  $0A     UNITS DIGIT
030C 85 04              STA    TEMP
030E 8A                 TXA
030F 18                 CLC
0310 2A                 ROLA
0311 2A                 ROLA
0312 2A                 ROLA
0313 2A                 ROLA
0314 65 04              ADC    TEMP
0316 60                 RTS
```

```
                    SUBROUTINE  MESSAG

                    ENTRY  JSR MESSAG

                    THIS ROUTINE WILL PARADE THE MESSAGE SPECIFIED
                    BY THE CALLER ACROSS THE LEDS.  THE A REGISTER
                    CONTAINS THE LO BYTE OF THE MESSAGE ADDRESS.  THE
                    X REG. CONTAINS THE HI BYTE OF THE MESSAGE ADDRESS.
                    THE FIRST BYTE OF THE MESSAGE CONTAINS THE NUMBER
                    OF BYTES IN THE MESSAGE MINUS 5.  THIS COUNT
                    INCLUDES THE FIRST BYTE

0317 8D 24 03  MESSAG STA   MAD    +01 CHANGE INSTRUCTION
031A 8E 25 03         STX   MAD    +02
031D 8D 37 03         STA   MADX   +01 CHANGE INSTRUCTION
0320 8E 38 03         STX   MADX   +02
0323 AD FF FF  MAD    LDA   $FFFF  ADDRESS WILL BE CHANGED
0326 85 10            STA   CLIM
0328 A9 00            LDAIM $00
032A 85 0D            STA   COUNT
032C 85 0E            STA   LOOPA
032E 85 0F            STA   LOOPB
0330 E6 0D            INC   COUNT
0332 A4 0D     MESS   LDY   COUNT
0334 A2 00            LDXIM $00
0336 B9 FF FF  MADX   LDAY  $FFFF  ADDRESS WILL BE CHANGED
0339 9D 40 A6         STAX  DISBUF
033C C8               INY
033D E8               INX
033E E0 06            CPXIM $06
0340 D0 F4            BNE   MADX
0342 E6 0D            INC   COUNT
0344 20 06 89  MESSA  JSR   SCAND
0347 E6 0E            INC   LOOPA
0349 D0 F9            BNE   MESSA
034B E6 0F            INC   LOOPB
034D A5 0F            LDA   LOOPB
034F C9 02            CMPIM $02
0351 D0 F1            BNE   MESSA
0353 A5 0E            LDA   LOOPA
0355 85 0F            STA   LOOPB
0357 A5 0D            LDA   COUNT
0359 C5 10            CMP   CLIM
035B D0 D5            BNE   MESS
035D 60               RTS

                    THE FAILURE MESSAGE BEGINS AT LOCATION 0360.
                    THE FIRST BYTE IS THE HEX NUMBER OF BYTES IN
                    THE MESSAGE MINUS FIVE.  THE MESSAGE IS IN THE
                    FORM OF SEGMENT CODES.  A MEMORY LISTING FOLLOWS.
                    LOAD THIS BEGINNING AT LOCATION 0360.

                    0360  0B 00 00 6E 3F 3E 00 38 3F 3F
                    0368  3F 3F 6D 79 00 00 00 00

                   THE SUCCESS MESSAGE BEGINS AT LOCATION 0380.

                    0380  08 00 00 39 5C 50 50 79
                    0388  58 78 00 00 00
```

SYM-1 TAPE DIRECTORY

John Gieryic
2041 138th Avenue N.W.
Andover, MN 55303

The SYM-1's high speed tape format enables recording and loading of 1K of RAM in just a few seconds (185 bytes per second). This quick and easy means of saving and restoring memory will have you SYM-1 owners quickly wrapped up in tape. With the possibility of 254 ID's (01 thru FE) you may forget which ID's you've already used or where you stored a particular identifier. Maintaining records sometimes seems secondary when you are eagerly pursuing an idea.

This program will refresh your memory quickly. When DIRECTORY "finds" a tape record it will extract the ID, startind address and ending address + 1. This information will be paraded across the LED's in much the same format used when you saved the data on tape. The program will then continue its search for more records. The process is terminated by pressing the RST key.

The first part of the program (locations 205 thru 232) is taken from the monitor routine LOADT. Since this is not a subroutine (callable by a JSR), I had to copy the necessary logic into my program. The last part of the program makes extensive use of subroutine calls to two of my own subroutines and several of the monitor's. Any newcomers to programming should take time to trace through this in order to see the power of subroutines.

SYM TAPE DIRECTORY

High Speed Format Only: START: GO 200 CR

TAPE FORMAT:

256 Sync Char * ID SAL SAH EAL+1 EAH+1

DATA / CKL CKH EOT EOT

This program will extract the tape identifier (ID), the starting address (SAL and SAH), the ending address (EAL and EAH) and will "parade" this information on the LED's. The program will then go back to the tape and search for the next record. The program is terminated by pressing the RST key.

```
                        SYM TAPE DIRECTORY

                        SYM REFERENCES

                        ACCESS *     $8B86
                        START  *     $8DB6
                        SYNC   *     $8D82
                        RDCHTX *     $8DDE
                        RDBYTX *     $8E28
                        RDBYTH *     $8DE2
                        OUTDSP *     $89C1
                        NIBASC *     $8309
                        SCAND  *     $890B
                        DISBUF *     $A641

                        DDRIN  *     $A002
                        VIAACR *     $A00B
                        LATCHL *     $A004
                        MODE   *     $00FD

                               ORG   $0000

0000 00                 ID     =     $00    TAPE ID LOCATION
0001 00                 SAL    =     $00
0002 00                 SAH    =     $00
0003 00                 EAL    =     $00
0004 00                 EAH    =     $00
0005 00                 TEMP   =     $00
0006 00                 LCNT   =     $00    LOW LOOP COUNTER
0007 00                 HCNT   =     $00    HIGH LOOP COUNT
```

```
0200                    ORG   $0200  PROGRAM ORIGIN

0200 20 86 8B  BEGIN    JSR   ACCESS ENABLE SYM PROTECTED MEMORY
0203 A0 80              LDYIM $80    SET HIGH SPEED MODE
0205 20 B6 8D           JSR   START  INIT TAPE ROUTINES
0208 AD 02 A0           LDA   DDRIN
020B 29 BF              ANDIM $BF
020D 8D 02 A0           STA   DDRIN
0210 A9 00              LDAIM $00
0212 8D 0B A0           STA   VIAACR
0215 A9 1F              LDAIM $1F    SET UP TIMER
0217 8D 04 A0           STA   LATCHL
021A 20 82 8D  FIND     JSR   SYNC   SEARCH TAPE FOR RECORD
021D 20 DE 8D  READ     JSR   RDCHTX GET CHARACTER
0220 C9 2A              CMPIM '*     COMPARE FOR ASTERISK
0222 F0 06              BEQ   TEST   MATCH
0224 C9 16              CMPIM $16    TEST SYNC CHAR
0226 D0 F2              BNE   FIND
0228 F0 F3              BEQ   READ

022A A5 FD     TEST     LDA   MODE
022C 29 BF              ANDIM $BF
022E 85 FD              STA   MODE
0230 20 28 8E           JSR   RDBYTX GET ID
0233 85 00              STA   ID     SAVE ID
0235 20 28 8E           JSR   RDBYTX GET SAL FROM TAPE
0238 85 01              STA   SAL    SAVE
023A 20 28 8E           JSR   RDBYTX GET SAH FROM TAPE
023D 85 02              STA   SAH    SAVE
023F 20 E2 8D           JSR   RDBYTH GET EAL
0242 85 03              STA   EAL    SAVE
0244 20 E2 8D           JSR   RDBYTH GET EAH
0247 85 04              STA   EAH    SAVE
0249 A9 00              LDAIM $00    CLEAR OUT DISPLAY BUFFER
024B 8D 41 A6           STA   DISBUF
024E 8D 42 A6           STA   DISBUF +01
0251 8D 43 A6           STA   DISBUF +02
0254 8D 44 A6           STA   DISBUF +03
0257 8D 45 A6           STA   DISBUF +04
025A A5 00              LDA   ID     TAPE ID
025C 20 96 02           JSR   DISPL  SEND IT TO DISPLAY
025F A9 2D              LDAIM '-     ASCII DASH
0261 20 C1 89           JSR   OUTDSP SEND IT TO DISPLAY
0264 20 B5 02           JSR   DELAY  PAUSE
0267 A5 02              LDA   SAH    START ADDRESS HIGH
0269 20 96 02           JSR   DISPL  SEND TO DISPLAY
026C A5 01              LDA   SAL    START ADDRESS LOW
026E 20 96 02           JSR   DISPL  SEND TO DISPLAY
0271 A9 2D              LDAIM '-     DASH
0273 20 C1 89           JSR   OUTDSP DISPLAY IT
0276 20 B5 02           JSR   DELAY  PAUSE
0279 A5 04              LDA   EAH    END ADDRESS HIGH
027B 20 96 02           JSR   DISPL
027E A5 03              LDA   EAL    END ADDRESS LOW
0280 20 96 02           JSR   DISPL
```

```
0283 A9 00          LDAIM $00    ADD 2 TRAILING BLANKS
0285 20 C1 89       JSR   OUTDSP
0288 20 B5 02       JSR   DELAY
028B A9 00          LDAIM $00
028D 20 C1 89       JSR   OUTDSP
0290 20 B5 02       JSR   DELAY
0293 4C 00 02       JMP   BEGIN  GO TO NEXT RECORD ON TAPE
```

SUBROUTINE DISPL

ENTRY LDA (BINARY DATA)
JSR DISPL

THE UPPER FOUR BITS IN THE A REGISTER ARE CONVERTED TO THEIR ASCII EQUIVALENT, SENT TO THE DISPLAY VIA SUBROUTINE DELAY. NEXT THE PROCESS IS REPEATED WITH THE LOWER FOUR BITS.

```
0296 85 05     DISPL  STA   TEMP   SAVE A REGISTER
0298 6A               RORA         RIGHT JUSTIFY LEFT FOUR BITS
0299 6A               RORA
029A 6A               RORA
029B 6A               RORA
029C 29 0F            ANDIM $0F    MASK TO FOUR BITS
029E 20 09 83         JSR   NIBASC CONVERT TO ASCII
02A1 20 C1 89         JSR   OUTDSP SEND TO DISPLAY
02A4 20 B5 02         JSR   DELAY  PAUSE
02A7 A5 05            LDA   TEMP   RESTORE A
02A9 29 0F            ANDIM $0F    MASK OFF TO LOWER FOUR BITS
02AB 20 09 83         JSR   NIBASC CONVERT TO ASCII
02AE 20 C1 89         JSR   OUTDSP SEND TO DISPLAY
02B1 20 B5 02         JSR   DELAY  PAUSE
02B4 60               RTS          RETURN
```

SUBROUTINE DELAY

ENTRY JSR DELAY

THIS ROUTINE WILL CALL SCAND FOR A PERIOD OF TIME IN ORDER TO ILLUMINATE THE 6 LED'S

```
02B5 A9 00     DELAY  LDAIM $00    INIT LOOP COUNTERS
02B7 85 06            STA   LCNT
02B9 85 07            STA   HCNT
02BB 20 06 89  WAIT   JSR   SCAND  SYM DISPLAY
02BE E6 06            INC   LCNT
02C0 D0 F9            BNE   WAIT   DELAY
02C2 E6 07            INC   HCNT
02C4 A5 07            LDA   HCNT   TEST COUNTER
02C6 C9 03            CMPIM $03
02C8 D0 F1            BNE   WAIT
02CA 60               RTS
```

SYM-1 6522-BASED TIMER

John Gieryic
2041 138 Avenue, NW
Andover, MN 55303

Your SYM-1 comes with a number of timers capable of a wide range of timing intervals. Unfortunately the SYM REFERENCE MANUAL does not provide information which can easily be digested by a novice. I'd like to attempt a more down to earth description of timer 1 on the **Versatile Interface Adapter 6522** for those of us who aren't hardware inclinded. This timer is capable of very accurate time delays in the range of fractions of a second. It has an interrupt associated with it plus the ability to generate evenly spaced interrupts.

Setting Up The Interrupts

The first step in programming this timer is to place an address in the **Interrupt Request Vector [IRQ]** located at address A67E and A67F. A67E contains the low byte of the address and A67F contains the high byte. This address in the **IRQ** is the location you will be "jerked to" when the timer times down and generates an interrupt. Your code will be as follows:

Location	Code		
200	20 86 8B	JSR ACCESS	disable memory write protect
203	A9 00	LDA #00	interrupt address
205	8D 7E A6	STA A67E	Low byte
208	A9 03	LDA #03	
20A	8D 7F A6	STA A67F	High byte

Our next step is to set two locations so the hardware can "see" the interrupt and tell us where it is coming from. These two locations are the **Interrupt Flag Register [IFR]** at location A00D and the **Interrupt Enable Register [IER]** at location A00E. The **IER** controls interrupts from 7 different sources on the **6522**. We will only be interested in bit 6. This is the one for our timer T1. We must set this bit to a logic 1. This tells the **6522** we will accept interupts from timer T1. The code follows:

Location	Code	
20D	A9 CO	LDA #CO
20F	8D OE AO	STA AOOE

"Hey, wait a minute! Where did that 'C' come from? I thought you said we were only going to set bit 6?"

Yes, I did. We must supply the **6522** with a bit more information (no pun intended). We must tell it we are going to **SET** one of the **IER** bits. This is done by setting bit 7 to a logic 1, hence our CO. Note bits 0 thru 5 are a zero. This tells the **6522** we don't want to change the condition of any of the other bits in the **IER** when we do our store. From this you should be able to see how we **CLEAR** any one of the **IER** bits. You guessed it. Bit 7 will be a logic zero and the **IER** bit(s) to be cleared will be a logic 1.

The **Interrupt Flag Register [IFR]** tells the user which interrupt has occurred (when we get one). This information can be used by the interrupt routine to "see" which element on the **6522** gave us the interrupt. We want to initialize (clear) our flag bit for timer T1 (bit 6). I don't want to disturb any of the other bits. Note clearing a bit in the **IFR** is **not** the same as in the **IER**.

Location	Code	
212	AD OD AO	LDA AOOD
215	29 BF	AND #BF
217	8D OD AO	STA AOOD

When we do get an interrupt from any of the enabled **6522** devices (bit=1 in the **IER**) then bit 7 in the **IFR** and the corresponding bit in the **IFR** will both be set to a logic 1. We can determine if this interrupt came from the **6522** by just looking at bit 7 of the **IFR** (ASL followed by a test of the C bit). If bit 7 is a logic zero then the interrupt came from some other place. This will save some time when we are trying to find out where this interrupt originated. You should log this bit 7 information in the back of your mind since I won't use it here.

Setting Up The Timer

One more step before starting our timer. I'm going to set our timer to the free running mode. This means it will count down, give an interrupt and then immediately begin counting down again. I won't need to worry about instruction cycle times within any timing loops. I know I will get repeated interrupts at the exact interval requested. Setting the **Auxiliary Control Register [ACR]** bit 7 to a logic 1 establishes the free running mode.

Location	Code	
21A	A9 CO	LDA #CO
21C	8D OB AO	STA AOOB

Now we have the four mechanical steps finished...setting up the **IRQ, IFR, IER** and **ACR**. Setting the time delay is next. The T1 timer has two latches (high and low order) and two counters (high and low order). This results in a 16 bit counter. The low order latch is loaded first. In this example I will set up for a delay of .05 seconds. This corresponds to a count of C350 (one count for each microsecond).

Location	Code		
21F	A9 50	LDA #50	load low order latch
221	8D 06 AO	STA A006	

Now we will load the high order latch with the value C3. This instruction will do more than load the high order latch. It will also write the high order latch into the high order counter as well as write the low order latch into the low order counter. This one instruction will transfer all 16 bits from the latches to the counter at the same instant. Without this hardware assist we would be unable to load the counter accurately since the counter begins to count down immediately after being loaded.

Location	Code		
224	A9 C3	LDA #C3	load high order latch
226	8D 05 AO	STA A005	

The timer is now running and will generate an interrupt .05 seconds (C350) later. This corresponds to 50,000 clock cycles. If you were programming a clock your remaining code at location 229 would now initialize your hours, minutes and seconds counters, initialize the display buffer and then go into a tight loop calling SCAND in order to illuminate the LED's.

Servicing The Interrupt

Our interrupt routine at location 300 is now executed when we receive the interrupt. The first thing we must do is SAVE the processor status and registers. This is done so we can restore these items when we are finished with our interrupt processing and jump back into SCAND from where we were "jerked out."

Location	Code		
300	08	PHP	save processor status on stack
301	48	PHA	save accumulator on stack
302	8A	TXA	transfer X to A
303	48	PHA	save X register on stack
304	98	TYA	transfer Y to A
305	48	PHA	save Y register on stack

If you were programming a clock you would now increment a counter. If the counter equalled twenty then reset it and increment the time in the display buffer by one second.

Now the interrupt is "serviced." In order to clear the way for the next interrupt, the T1 interrupt flag must be reset otherwise the next interrupt will be blocked. This clearing can be done in either of two ways. Method 1 will write into the high order latch. This write uses a different address for the store instruction than the write used to initialize the timer counter. In doing this the T1 interrupt flag will be reset but it will not disturb the current value in the counter. Remember this is a free running counter in our example and automatically resets itself when the interrupt occurred. By this point in time it has already counted down from its original value of C350 toward zero (and the next interrupt). Method 2 will read the low order counter. Either method will reset the T1 interrupt flag.

Method 1

Code		
A9 C3	LDA #C3	
8D 07 AO	STA A007	

Method 2

Code		
AD 04 AO	LDA A004	

Now the processor status and registers can be restored and a return executed to the location in SCAND at which the interrupt occurred. Remember you **must** restore the registers in the exact reverse order used at the entrance to the interrupt routine. This is a major point.

Code		
68	PLA	pull accumulator from stack
A8	TAY	transfer to Y index
68	PLA	pull accumulator from stack
AA	TAX	transfer to X index
68	PLA	pull accumulator from stack
28	PLP	pull processor status from stack
40	RTI	Return from Interrupt

That's the end of the lesson for today. In a future article I will use the information presented here to develop an operating system for your SYM-1.

KIM-1 AS A DIGITAL VOLTMETER

Joseph L. Powlette and Charles T. Wright
Hall of Science, Moravian College
Bethlehem, PA 18018

Several programs have been described in the literature which turn a KIM-1 microcomputer into a direct reading frequency counter. In "A Simple Frequency Counter Using the KIM-1" by Charles Husbands (MICRO, No. 3, Pp. 29-32, Feb/Mar,1978) and in "Here's a Way to Turn KIM Into a Frequency Counter" by Joe Laughter (KIM User's Note Issue 3, Jan, 1977), good use is made of KIM-1's interval timers and decimal mode to produce a useful laboratory instrument. A simple change in hardware will allow these same programs to serve as the basis of a direct reading digital voltmeter. This article describes an inexpensive voltage-to-frequency converter (VFC) circuit which is compatible with these programs and also describes some software modifications which will allow Husbands' program to operate down to low frequency (10 HZ) values.

Hardware Configuration

The VFC circuit is shown in Figure 1. The 4151 chip is manufactured by Raytheon and is available from Active Electronic Sales Corp., P.O. Box 1035, Framingham, MA 01701 for $5.00 or from Jameco Electronics, 1021 Howard Street, San Carlos, CA 94070 for $5.95. The circuit parameters given in Figure 1 have been modified from the values suggested by the manufacturer in order to match the pulse requirement for the KIM $\overline{IRQ}$ signal. The frequency of the output pulse is proportional to the input voltage and the 1K (multiturn) trimpot is used to adjust the full-scale conversion so that 10 volts corresponds to a frequency of 10 KHz. It is not necessary to calibrate the KIM-1 as a frequency meter since any variation in its timing can be compensated for by the trimpot. A known potential is connected to the VFC input and the trimpot adjusted until the KIM readout agrees with the known voltage value. The linearity of the VFC is better than 1% down to 10 mv (linearity of 0.05% can be achieved in a "precision mode" which is described in the Raytheon literature). The circuit will not respond to negative voltages and protection of the chip is provided by the 1N914 diode. If negative voltage readings are also required, the input to the VFC can be preceded by an absolute value circuit (see IC OP-AMP cookbook by Jung, p. 193, Sams Pub.).

To operate the system using Laughter's software the following connections should be made: 1) the output (pin 3) of the VFC to the PB0 input of KIM (pin 9 on the application connector) and 2) PB7 on the KIM to $\overline{IRQ}$ on the KIM (A-15 to E-4). Execution of the program should cause the voltage to flash on the KIM display in one second intervals.

The software described in Husbands' article will not operate below 500 Hz. This limit is caused by the fact that the contents of the interval timer are read to determine if the 100 millisecond interval has elapsed and since the interval counter continues to count (at a 1T rate) after the interval has timed out, there are times when the contents of the interval timer are again positive. If the interrupt should sample during this time, the branch on minus instruction will not recognize that the interval has elapsed. This problem will manifest itself as a fluctuating value in the display and is most likely to occur at low frequencies. One solution is to establish the interval timer in the interrupt mode and then allow the program to arbitrate the interrupt, i.e., to determine whether the interrupt was due to the input pulse or the expiration of the 100 millisecond interval timer. The necessary changes to Husbands' program are given in Figure 2. The hardware connections are: 1) output of the VFC (pin 3) to the KIM $\overline{IRQ}$ (pin 4 on the KIM expansion connector), and 2) PB7 on the KIM to $\overline{IRQ}$ on the KIM (A-15 to E-4). The modified program starts at 0004 with a clear interrupt instruction. Locations 17FE and 17FF should contain 21 00 and 17FA and 17FB should have values 00 10 (or 00 1C).

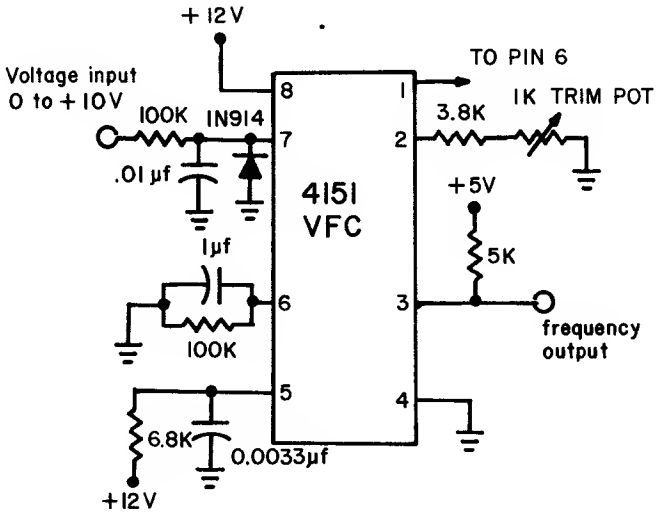


Figure 1. Voltage-to-Frequency Converter (VFC) circuit.

Additional Comments

The program modifications above will also extend Husbands' frequency counter circuit down to 10 Hz (corresponding to 1 input. interrupt in 100 milliseconds). Since the 74121 monostable multivibrator does not have an open collector output, PB7 should not be connected (along with the 74121 output) directly to the KIM IRQ. Two solutions are:

1. Leave PB7 unconnected. The expiration of the 100 millisecond clock will be recognized on the next input interrupt after the timer has timed out. The interval timer will not interrupt the microprocessor, however.

2. Connect PB7 to one input of a two input AND gate and the output of the monostable to the second input. The output of the AND gate should be connected to the KIM IRQ. The expiration of the 100 millisecond interval will now also interrupt the processor and will result in a faster response to a change in frequency values (from high to very low) as well as a more accurate low frequency count.

The authors would like to thank Charles Husbands for taking the time to answer our questions and for pointing out the article by Laughter.

```
                    ORG  $  0004

0004    58          CLI       clear interrupt flag
.
0014    8D 0F 17    STA       clock in interrupt mode
.
0024    AD 07 17    LDA       read interrupt flag bit 7
.
003C    8D 0F 17    STA       clock in interrupt mode
.
```

Figure 2. Changes in Husbands' program to extend the low frequency range to 10 Hz.

INSIDE THE KIM TTY SERVICE

Ben Doutre
621 Doyle Road
Mont St-Hilaire, Quebec
Canada J34 1M3

The fact the KIM's serial TTY port, plain and unmodified, will operate comfortable at 9600 bauds does not seem to be widely known. I, for one, went the parallel interface route as soon as I acquired a higher speed terminal, and I suspect that many others may have done likewise. After all, what can one expect of an interface described in the User's Manual in these terms: "You are not restricted to units with specific bit rates (10 CPS for TTY) since the KIM-1 system automatically adjusts for a wide variety of data rates (10 CPS. 15 CPS. 30CPS. ETC.(.' "That's pretty wide, alright, from 10 to etc. Other writers have been equally vague. Gary Tater in MICRO 9:14, "A Fast Talking TIM" mentions that "KIM can adapt to terminal frequencies up to 2400 baud...". This was the last straw, and I either had to pull the plug on my "Fast Talking KIM". or attempt to put the record straight

First off, let me say that according to my interpretation of what goes on in KIM, the theoretical maximum baud rate of the TTY port is 15,625. How's that for pinning down the etc? Not that you should try to operate at this rate without some of the well-known "fine tuning", but there is no reason why you can't hook up your 9600 or 4800 baud terminal, with 30 cents worth of gates, and be up and running, with or without reading the following details. If you want to know from whence this bonanza, here is the story.

The smarts for the KIM TTY interface are in the monitor software, so let's start at that end. There are two main TTY I/O routines: GETCH at 1E5A and OUTCH at 1EA0. GETCH returns with the character in A but strips off the parity bit in the process. If you need bit 7 (counting from 0) for your own deep, dark reasons, then retrieve the full character from CHAR at OOFE on your return. OUTCH (love that label!) outputs a stop bit, then a start bit, then 8 data bits (LSB first), then another stop bit. It may seem illogical to start with a stop, but remember that, aside from slow machinery, the main purpose of a stop bit (line high) is to make sure that the start bit (line low) will be recognized. In any case, the stop interval is 2 bits long plus the delay between calls to OUTCH.

Both GETCH and OUTCH are timed by subroutine DELAY at IED4. (GETCH also used DEHALF to move its strobe to the mid-point of a bit interval, but let's not get technical.) DELAY does its thing based on the contents of a 16-bit counter named, for some obscure reason, CNTH30 (high byte, at 17F3) and CNTL30 (low byte, 17F2). If this counter is equal to 0000 or less, DELAY falls through all the way, with a resulting minimum bit time of 64us. (Let's assume your crystal is bang-on 1 MHz.) Presto: devide 64us into a million, and you come up with 15,625 baud.

Not convinced? OK, here's more. Every time we add one to the counter, DELAY adds another 14 us to its timing loop. The high end of the baud scale looks like this:

Counter	Bit Time (us)	Baud Rate
0000	64	15,625
0001	78	12,820
0002	92	10,869
0003	106	9,434
0004	120	8,333

If we turn this around and start with some of the usual standard baud rates, we can calculate the bit times and counter values required. For instance, 9600 bauds obviously needs something betweem 2 and 3. DELAY doesn't do fractions - it doesn't even like odd numbers. And how does the counter get properly loaded anyway'?

We've left the best to the last, a little jewel called DETCPS at 1C2A. DETCPS is entered following a system reset with TTY enabled. Its brief hour of glory is in measuring the duration of the start pulse of the first character you feed in after a Reset. It quickly stuffs the results in the 16-bit counter, then goes out for coffee until the next Reset. The question is: will DETCPS buy 9600 bauds? The answer is YFS. albeit a little reluctantly. The thing is the DETCPS is sampling the input port, waiting for the line to go low - it checks for this every 9 us, so it could miss your start pulse start by this much. Once the line is low, it squirrels away 14 us counts, checking for line high every 14 us. So it could miss the end of your start pulse by 14 us.

At 10, 15, 30 or etc CPS, this sloppiness is probably acceptable. With a Model 33 on the line, DETCPS gaily reports 02C2 plus/minus OB, for instance. But if it comes up with 0004 instead of 0003 at 9600 bauds, your TV screen will give you a reasonable facsimile of a Chinese fortune cookie slip. Just look at it as another Butterfield game - Reset-Delete-Reset-Delete-Reset-Delete BINGO! Anyway, how many times a day do you Reset? Once you get that 3, your link with KIM will be rock solid.

There are a number of facinating details, but I will spare you the pyrotechnics. If all this is on the leve, I should be able to prove it, right? Well, I have an ESAT-100 (RHS Marketing) video board equipped with an AY3-1015 UART hooked up to the KIM TTY port. The manual admits to a -1% to DETCPS. I set the speed selector switch to each of the 6 rates available, did 10 resets at each and recorded the counts. (A clever piece of programming, at that!) Except for 9600, all resets were OK the first time around. The counts did not vary, except for 300 baud. The results look like this:

Baud Rate	Bit Time (us)	Calc. Count	Meas'd Count
9600	104.2	0003	0003
4800	208.3	000A	000B
2400	416.7	0019	001A
1200	833.3	0037	0038
600	1666.7	0072	0074
300	3333.3	00FA	00EC/00FD

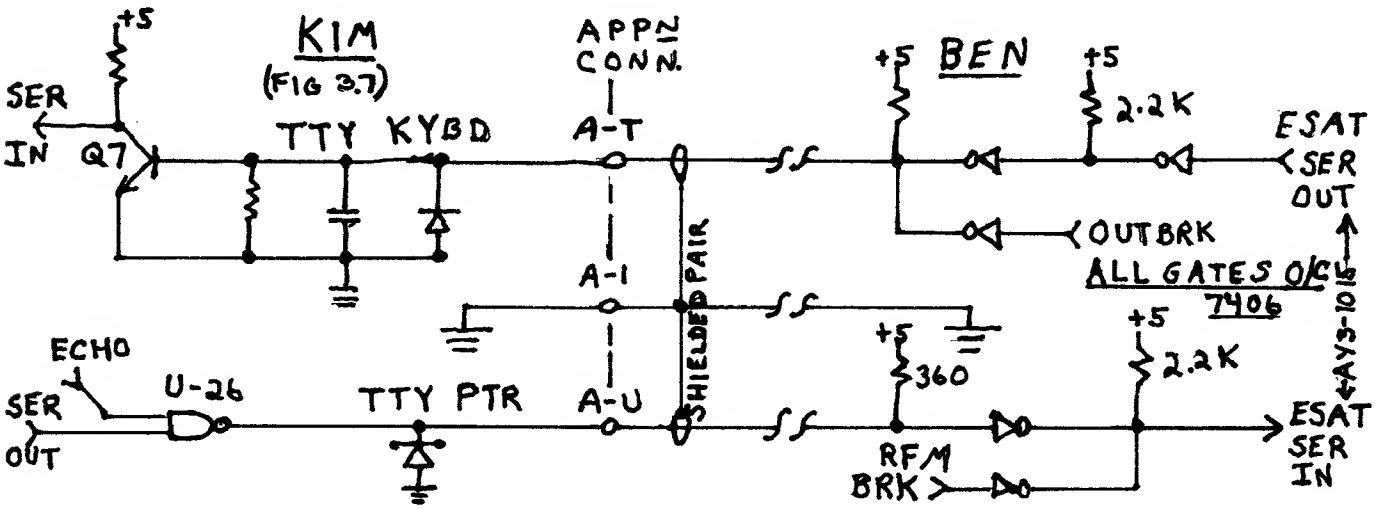
A few further words of explanation for the fellow who may be hung up because he has been spared intimate relations with "real" TTY machines. (You experts can go figure out an algorithm or two - try infinite recursion on "Every rule has an exception, except this one.")

Referring to the KIM-1 User's Manual, Fig. 3.7, you will see two KYBD lines and two PTR lines. The action at the other end of these lines is assumed to be as follows: - During idle conditions, the keyboard lines are shorted out, generating a continuous high at the input to Q7; the printer lines are connected to a "selector magnet" (quaint) or a relay which is drawing a nominal 20 mA. -when the keyboard is sending characters, the KYBD lines are open-circuited for zero bits and shorted for one bits. When KIM sends characters on the PTR lines, it opens the circuit for zero bits by floating the output of O/C gate U26 (7438), and closes the circuit for one bits by pulling U26 to ground. Incidentally, this 7438 can sink up to 48 mA.

If you want to simulate this hardware with some other device, you need to feed the line labelled "TTY KYBD" with positive logic signals (low for ones, open for zeros) from the line labelled "TTY PTR". You should note that the keyboard line has a 220-ohm pull down resistor on it, and that the printer line has no pull-up.

You may also notice, if your terminal has a FDX/HDX selector switch or jumper, that the FDX no longer works as advertised. This is just KIM trying to be helpful, with a wired-in interconnect which echos received characters on the output line. If this keeps you awake at night, cut the trace between pin 11 and U15 and pin 10 of U26, and connect pin 10 of U26 to Vcc. (I haven't tried it, but it should work. I'm a sound sleeper.)

If you need a for-example, I show a diagram of my own interface logic, based on a 7406 gate package, which is working quite satisfactorily. There are probably 1000 other ways of doing it, each one of which can be improved by SuperSilicon. If it works and doesn't smoke, have at it.



KIMBASE

Dr. Barry Tepperman
25 St. Mary St., No. 411
Toronto, Ontario M4Y 1R2
Canada

KIMBASE is an application program written in the 6502 microprocessor machine language, designed to make use of the monitor subroutines and memory configuration of the KIM-1 microcomputer, for conversion of unsigned integers from one base to another. The input integer (designated NUMBER is to be no greater than 6 digits in length; large 6-digit integers may cause overflow in the multiplication subroutines with consequent errors in conversion. The base to be converted from (designated BASE1) and to be converted to (BASE2) are each in the range from 02_H to 10_H; the lower limit is set by mathematical reality and the upper by the limited enumeration available from the KIM-1 keypad.

The program is started by placing NUMBER, lowest order byte last, in page zero 4C-4E, BASE1 (expressed in hexadecimal) in 4A, and BASE2 (also in hexadecimal) in 4B. The program starts at 0200, and will light up the KIM-1 LED display with either an error message (according to an error flag stored in zero page 02, called ERROR), or a result display with the input data and a final result up to 18_Hdigits in length (RESULT stored in 03-0E) in successive segments in a format to be discussed below, or a combination of both displays, in an endless loop until the RS key is pressed.

Program Function

After initialization of data workspace, several tests of input data validity are conducted. KIMBASE recognizes four error states:

a) NUMBER will remain same after conversion (i.e. NUMBER=00000x where x is less than either base). KIMBASE sets ERROR=01, RESULT=NUMBER, and shows both error and result displays.

b) Either or both bases are outside the permissable limits of $02-10_H$. KIMBASE resets bases under 02 to equal 02 and bases exceeding 10_H to equal 10_H, and executes program to display result without an error display.

c) BASE1=BASE2. KIMBASE sets ERROR=02, RESULT=NUMBER, and shows error and result displays.

d) NUMBER enumeration is impermissable, as one or more digits =BASE1 (e.g., attempting NUMBER=1C352A with BASE1=05). KIMBASE sets ERROR=03, shows error display, and aborts further execution.

Note that error states "a" and "c", above, are not mutually exclusive, and that KIMBASE sets the error flag ERROR and goes to the appropriate response routine after only one positive test. Errors are displayed as a continuous flashing LED readout "ErrorY" where Y=ERROR.

Following the test routines, if BASE1$\neq 10_H$, KIMBASE converts NUMBER into its hexadecimal equivalent by successive generation of powers of BASE1, multiplication of the appropriate power by the individual digits of NUMBER (remapped by masking and shifting into array N), and successive addition of all the hexadecimal products. This intermediate result is placed in array HEXCON. A successive loop algorithm was used for multiplication rather than a shift-and-binary-add algorithm for economy of coding.

$$\text{HEXCON} = \left[\sum_{y=1-6} N(Y) * \text{BASE1}^{(y-1)} \right]_{10}$$

This calculation is bypassed and NUMBER entered directly into HEXCON if BASE1=10_H.

After the conversion to hexadecimal, if BASE2=10_H, KIMBASE sets RESULT=HEXCON and the result display is initiated. If BASE2$\neq 10_H$, HEXCON is converted into BASE2 by the common successive division procedure by BASE2 with mapping of remainders through an intermediate array into RUSULT.

Results are displayed on the KIM-1 6-digit display as successive 1-second displays of NUMBER, BASE1 and BASE2, and RESULT divided into 6-digit segments, in the format:

```
NNNNNN   (NUMBER1-NUMBER3)
IIbbOO   (II=BASE1; OO=BASE2)
RRRRRR   (RESULT1-RESULT3)
RRRRRR   (RESULT4-RESULT6)
RRRRRR   (RESULT7-RRSULT9)
RRRRRR   (RESULTA-RESULTC)
```

which loops endlessly. Where ERROR=01 or 02, the error message precedes the result display, and loops endlessly in the display.

All intermediate arrays and products have been retained in the zero page data workspace to facilitate any debugging or further elaboration of the program that other users may find necessary.

Users of non-KIM 6502-based microcomputers may implement KIMBASE easily with appropriate relocation of program and workspace (if necessary) and replacement of the display subroutines (SHOWER-TIMER1, SHORES-TIMER2) with appropriate machine-dependant output routines (or by BRK instructions with manual interrogation of the appropriate arrays to determine output).

KIMBASE - MAIN PROGRAM LISTING

```
*************** this section initializes data workspace and constants **********

            CLD                 0200    D8          select binary mode
            LDX  $#48           01      A2  48      set workspace byte counter
ZERO1       LDA  $#00           03      A9  00
            STA  ARRAY,X        05      95  01      zero common workspace
            DEX                 07      CA          decrement counter
            BNE  ZERO1          08      D0  F9      if ≠0 loop back
            LDA  $#0F           0A      A9  0F
            STA  MASK1          0C      85  0F      set MASK1=0F
            LDA  $#F0           0E      A9  F0
            STA  MASK2          10      85  10      set MASK2=F0
```

```
                LDA  $#Ø5                12    A9  Ø5
                STA  PWR                 14    85  ØØ       set PWR=Ø5
                LDX  $#FF                16    A2  FF
                TXS                      18    9A           set stack pointer=FF

*************** this section tests input data validity ************************

TST1NR          LDA  $#ØØ                19    A9  ØØ          TEST - ERROR STATE "a"
                CMP  NUMBER1             1B    C5  4C       NUMBER1=ØØ?
                BNE  TST1BS              1D    DØ  14       no? go to next test
                CMP  NUMBER2             1F    C5  4D       NUMBER2=ØØ?
                BNE  TST1BS              21    DØ  1Ø       no? go to next test
                LDA  NUMBER3             23    A5  4E
                CMP  BASE2               25    C5  4B       NUMBER3< BASE2?
                BCC  CORR1               27    9Ø  Ø3       yes? go to correction routine
                JMP  TST1BS              29    4C  33  Ø2   go to next test
CORR1           LDA  $#Ø1                2C    A9  Ø1
                STA  ERROR               2E    85  Ø2       set ERROR=Ø1
                JMP  CORR3A              3Ø    4C  5A  Ø2    and jump to CORR3A
TST1BS          LDX  $#Ø2                33    A2  Ø2          TEST - ERROR STATE "b"
TST1B2          LDA  BASE,X              35    B5  49
                CMP  $#Ø2                37    C9  Ø2       BASE(X) < Ø2?
                BCC  CORR2A              39    9Ø  ØB       yes? go to correction routine
                CMP  $#11                3B    C9  11       BASE(X)≥ 11?
                BCC  RESET1              3D    9Ø  ØB       no? bypass correction
CORR2B          LDA  $#1Ø                3F    A9  1Ø
                STA  BASE,X              41    95  49       otherwise set BASE(X)=1Ø
                JMP  RESET1              43    4C  4A  Ø2    and bypass next correction
CORR2A          LDA  $#Ø2                46    A9  Ø2
                STA  BASE,X              48    95  49       set BASE(X)=Ø2
RESET1          DEX                      4A    CA           decrement loop counter
                BNE  TST1B2              4B    DØ  E8       and go back if ≠Ø
TST2BS          LDA  BASE2               4D    A5  4B          TEST - ERROR STATE "c"
                CMP  BASE1               4F    C5  4A       BASE2=BASE1?
                BEQ  CORR3               51    FØ  Ø3       yes? go to correction routine
                JMP  TST3BS              53    4C  6A  Ø2   otherwise bypass
CORR3           LDA  $#Ø2                56    A9  Ø2
                STA  ERROR               58    85  Ø2       set ERROR=Ø2
CORR3A          LDX  $#Ø3                5A    A2  Ø3
                LDY  $#ØC                5C    AØ  ØC
CORR3B          LDA  NUMBER,X            5E    B5  4B       read NUMBER
                STA  RESULT,Y            6Ø    99  Ø2  ØØ    into RESULT
                DEY                      63    88           decrement counters
                DEX                      64    CA                    .............
                BNE  CORR3B              65    DØ  F7        and loop until complete
                JSR  SHOWER              67    2Ø  AØ  ØØ   display error message
TST3BS          LDA  BASE1             ØØ6A    A5  4A
                CMP  $#1Ø                6C    C9  1Ø       BASE1=1Ø?
                BCC  TST2NR              6E    9Ø  ØC       no? go to next test
                LDX  $#Ø3                7Ø    A2  Ø3
HEXMAP          LDA  NUMBER,X            72    B5  4B       yes? read NUMBER
                STA  HEXCON,X            74    95  25        into HEXCON
                DEX                      76    CA
                BNE  HEXMAP              77    DØ  F9          for all 3 bytes
                JMP  HEX1                79    4C  1F  Ø3   and bypass hex conversion
TST2NR          LDA  BASE1               7C    A5  4A          TEST - ERROR STATE "d"
                STA  BSTR1               7E    85  11       store BASE1
                ASL  ASL                 8Ø    ØA  ØA
                ASL  ASL                 82    ØA  ØA        and left shift 4 bits
                STA  BSTR2               84    85  12          to store BSTR2=(1Ø*BASE1)
                LDY  $#Ø2                86    AØ  Ø2
TLP2            LDX  $#Ø3                88    A2  Ø3
TLP1            LDA  NUMBER,X            8A    B5  4B       isolate each digit NUMBER(X)
                AND  MASK,Y              8C    39  ØE  ØØ    by masking
                CMP  BSTR,Y              8F    D9  1Ø  ØØ      and compare with BSTR
                BCC  TRESET              92    9Ø  Ø3       if less, reset loop
                JMP  CORR4               94    4C  AØ  Ø2   otherwise impermissable - correct
TRESET          DEX                      97    CA           decrement counter NUMBER
                BNE  TLP1                98    DØ  FØ       and repeat for corresponding digits
                DEY                      9A    88           decrement counter BSTR/MASK
                BNE  TLP2                9B    DØ  EB       and repeat for remaining digits
                JMP  REMAP               9D    4C  A7  Ø2   go to REMAP
CORR4           LDA  $#Ø3                AØ    A9  Ø3
                STA  ERROR               A2    85  Ø2       set ERROR=Ø3
                JSR  SHOWER              A4    2Ø  AØ  ØØ   and display error message
```

```
*************** this section remaps NUMBER for conversion to hex ***************

REMAP     LDX  $#Ø3          A7    A2  Ø3
REMAP1    LDA  NUMBER,X      A9    B5  4B       load NUMBER
          STA  NHI,X         AB    95  12        into NHI
          STA  NLO,X         AD    95  15         and into NLO
          DEX                AF    CA
          BNE  REMAP1        BØ    DØ  F7       loop until done
          LDX  $#Ø3          B2    A2  Ø3
MASKS1    LSR  NHI,X         B4    56  12       right shift
          LSR  NHI,X         B6    56  12        NHI
          LSR  NHI,X         B8    56  12         4 bits
          LSR  NHI,X         BA    56  12          .......
          LDA  NLO,X         BC    B5  15
          AND  MASK1         BE    25  ØF       isolate right digit NLO
          STA  NLO,X         CØ    95  15
          DEX                C2    CA
          BNE  MASKS1        C3    DØ  EF       loop until done
          LDY  $#Ø1          C5    AØ  Ø1
          LDX  $#Ø3          C7    A2  Ø3
REMAP2    LDA  NLO,X         C9    B5  15       store NLO into N
          STA  N,Y           CB    99  18  ØØ
          INY                CE    C8            alternately
          LDA  NHI,X         CF    B5  12         with NHI
          STA  N,Y           D1    99  18  ØØ      and in inverse order
          INY                D4    C8
          DEX                D5    CA
          BNE  REMAP2        D6    DØ  F1       loop until done

*************** this section converts N into hexadecimal ***************************

HEXCNV    LDY  $#Ø6          Ø2D8  AØ  Ø6        for six places
LP1PWR    JSR  PWRGEN          DA  2Ø  6Ø  ØØ     generate powers of BASE1
          LDA  N,Y             DD  B9  18  ØØ
          CMP  $#Ø1            EØ  C9  Ø1       N(Y)=Ø1?
          BEQ  RESET3          E2  FØ  ØB       if equal, go to RESET3
          BCC  RESET5          E4  9Ø  15       if less, go to RESET5
          STA  MULTP           E6  85  1F       set MULTP=N(Y)
RESET2    TYA                  E8  98           put index Y into accumulator
          PHA                  E9  48            and push onto stack
          JSR  MULT            EA  2Ø  8Ø  ØØ    multiply power by N(Y)
          PLA                  ED  68           pull accumulator from stack
          TAY                  EE  A8            and restore to Y
RESET3    CLC                  EF  18
          LDX  $#Ø3            FØ  A2  Ø3
RESET4    LDA  MULTC,X         F2  B5  1F       add new product
          ADC  HEXCON,X        F4  75  25        to intermediate product
          STA  HEXCON,X        F6  95  25         and store as intermediate product
          DEX                  F8  CA
          BNE  RESET4          F9  DØ  F7       loop until done
RESET5    DEY                  FB  88           for next place
          BEQ  HEX1            FC  FØ  21        if counter=Ø bypass
          DEC  PWR             FE  C6  ØØ       reduce power to be generated
          LDA  PWR           Ø3ØØ  A5  ØØ
          CMP  $#Ø1            Ø2  C9  Ø1       PWR=Ø1?
          BEQ  RESET6          Ø4  FØ  Ø2       yes? go to RESET6
          BCS  LP1PWR          Ø6  BØ  D2       greater? loop back to new conversion
RESET6    LDA  N,Y             Ø8  B9  18  ØØ
          STA  MULTC3          ØB  85  22       set MULTC=N(Y)
          LDA  $#ØØ            ØD  A9  ØØ
          STA  MULTC1          ØF  85  2Ø
          STA  MULTC2          11  85  21
          LDA  BASE1           13  A5  4A
          STA  MULTP           15  85  1F       set MULTP=BASE1
          LDA  PWR             17  A5  ØØ
          CMP  $#Ø1            19  C9  Ø1       PWR=Ø1?
          BEQ  RESET2          1B  FØ  CB       yes? go to RESET2
          BCC  RESET3          1D  9Ø  DØ       less? go to RESET3

*************** this section produces result from HEXCON when BASE2=1Ø *************

HEX1      LDA  BASE2           1F  A5  4B
          CMP  $#1Ø            21  C9  1Ø       BASE2=1Ø?
          BCC  ZERO2           23  9Ø  1Ø       no? go to ZERO2
          LDY  $#ØC            25  AØ  ØC
          LDX  $#Ø3            27  A2  Ø3
```

```
HEX2    LDA  HEXCON,X      29   B5 25        store HEXCON
        STA  RESULT,Y      2B   99 Ø2 ØØ     into RESULT
        DEY                2E   88
        DEX                2F   CA
        BNE  HEX2          3Ø   DØ F7        loop until done
        JSR  SHORES        32   2Ø 9Ø Ø3      and display result

********** this section divides HEXCON by BASE2 for crude conversion **********

ZERO2   STA  DIVIS         Ø335 85 2C        set DIVIS=BASE2
        LDX  $#Ø3          37   A2 Ø3
LPlDIV  LDA  HEXCON,X      39   B5 25        load HEXCON
        STA  DIVD,X        3B   95 28        into DIVD
        DEX                3D   CA
        BNE  LPlDIV        3E   DØ F9        loop until done
        LDY  $#18          4Ø   AØ 18        for 18H places
LP2DIV  JSR  DIVIDE        42   2Ø 1Ø Ø1     execute division
        LDA  RDR           45   A5 3Ø        load RDR
        STA  RSTOR,Y       47   99 3Ø ØØ     into RSTOR
        LDX  $#Ø2          4A   A2 Ø2
TSTlQO  LDA  QUO,X         4C   B5 2C
        CMP  $#Ø1          4E   C9 Ø1        QUO(1 or 2)≥Ø1?
        BCS  RESET7        5Ø   BØ Ø9        yes? go to RESET7
        DEX                52   CA
        BNE  TSTlQO        53   DØ F7        loop until done
        LDA  QUO3          55   A5 2F
        CMP  DIVIS         57   C5 2C        QUO3=DIVIS?
        BCC  ENDDIV        59   9Ø 15        less? go to ENDDIV
RESET7  LDX  $#Ø3          5B   A2 Ø3
RST7A   LDA  QUO,X         5D   B5 2C        load QUO
        STA  DIVD,X        5F   95 28        into DIVD
        LDA  $#ØØ          61   A9 ØØ
        STA  QUO,X         63   95 2C        zero QUO
        DEX                65   CA
        BNE  RST7A         66   DØ F5        loop until done
        STA  RDR           68   85 3Ø        zero RDR
        DEY                6A   88           decrement place counter
        BEQ  ENDV2         6B   FØ Ø9        if =Ø go to ENDV2
        JMP  LP2DIV        6D   4C 42 Ø3     otherwise back to divide routine
ENDDIV  DEY                7Ø   88           decrement place counter
        LDA  QUO3          71   A5 2F        load QUO3
        STA  RSTOR,Y       73   99 3Ø ØØ     into next RSTOR slot

********** this section maps RSTOR into RESULT for final result **********

ENDV2   LDY  $#ØC          76   AØ ØC
        LDX  $#18          78   A2 18
        CLC                7A   18
REMAP3  DEX                7B   CA
        LDA  RSTOR,X       7C   B5 30        left shift alternate bytes
        ASL  ASL           7E   ØA ØA        RSTOR 4 bytes
        ASL  ASL           8Ø   ØA ØA        ..............
        INX                82   E8
        ADC  RSTOR,X       83   75 3Ø        add to next byte RSTOR
        STA  RESULT,Y      85   99 Ø2 ØØ     and store as RESULT
        DEY                88   88
        DEX                89   CA
        DEX                8A   CA
        BNE  REMAP3        8B   DØ EE        loop until done
        JSR  SHORES        8D   2Ø 9Ø Ø3      and display result
```

1. PWRGEN

Subroutine to generate a^b by successive iterations of multiplication subroutine MULT with resetting of counters and intermediate products; allows unsigned binary or decimal arithmetic in 65Ø2 instruction set; maximum result memory allocated 18_H bits.

Requires: subroutines: MULT ØØ8Ø-Ø09B

data arrays: BASEl ØØ4A
PWR ØØØØ
PWRS ØØØ1
MULTP ØØ1F
MULTC ØØ2Ø-ØØ22

Inapplicable to PWR=ØØ,Ø1; calling program must test and bypass.

```
PWRGEN   LDA  PWR        ØØ6Ø   A5 ØØ    load power
         STA  PWRS         62   85 Ø1    store in counter
         DEC  PWRS         64   C6 Ø1    decrement counter
         LDA  BASE1        66   A5 4A
         STA  MULTP        68   85 1F    set multiplier=base
         STA  MULTC3       6A   85 22    set multiplicand=base
         LDA  $#ØØ         6C   A9 ØØ
         STA  MULTC1       6E   85 2Ø    zero 2 high-order bytes
         STA  MULTC2       7Ø   85 21     of multiplicand
         TYA               72   98       transfer index Y to accumulator
         PHA               73   48        and onto stack
MULTCL   JSR  MULT         74   2Ø 8Ø ØØ jump to MULT
         DEC  PWRS         77   C6 Ø1    decrement counter
         BNE  MULTCL       79   DØ F9    if ≠Ø return to MULTCL
         PLA               7B   68       pull accumulator from stack
         TAY               7C   A8        and restore to index Y
         RTS               7D   6Ø       return to main program
```

2. MULT

Subroutine multiplies 24-bit number (MULTC) by 8-bit number (MULTP) to yield 24-bit final product (MULTC) by successive iterations of nested addition loops. Intermediate product storage in MIDPRO. Allows unsigned decimal or binary operation in 6502 instruction set.

Requires : data arrays : MULTP ØØ1F
MULTC ØØ2Ø-ØØ22
MIDPRO ØØ23-ØØ25

Inapplicable to MULTP less than Ø2; calling program to test and bypass

```
MULT     LDY  MULTP        ØØ8Ø   A4 1F    loop counter=multiplier
         DEY                 82   88       decrement loop counter
         LDX  $#Ø3           83   A2 Ø3    set byte counter in loop
REDIST   LDA  MULTC,X        85   B5 1F    set intermediate register
         STA  MIDPRO,X       87   95 22     =multiplier
         DEX                 89   CA          for each byte in array
         BNE  REDIST         8A   DØ F9    loop until X=Ø
ADLP2    LDX  $#Ø3           8C   A2 Ø3    set byte counter in loop
         CLC                 8E   18       clear carry
ADLP1    LDA  MULTC,X        8F   B5 1F    add multiplicand
         ADC  MIDPRO,X       91   75 22     to intermediate product
         STA  MULTC,X        93   95 1F      store as new multiplicand
         DEX                 95   CA          for each byte in array
         BNE  ADLP1          96   DØ F7    loop until X=Ø
         DEY                 98   88       decrement loop counter
         BNE  ADLP2          99   DØ F1    another loop if Y≠Ø
         RTS                 9B   6Ø       return to main program
```

3. DIVIDE

Subroutine to divide 24-bit dividend (DIVD) by 8-bit divisor (DIVIS) to yield 24-bit quotient (QUO) and 8-bit remainder (RDR) by successive shift and subtraction processes; unsigned binary arithmetic only in 6502 instruction set. Intermediate quotient storage in QUO. Requires initialization of RDR and array QUO to Ø by calling program, DIVIS≠Ø.

Requires : data arrays : DIVD ØØ29-ØØ2B
DIVIS ØØ2C
QUO ØØ2D-ØØ2F
RDR ØØ3Ø

```
DIVIDE   LDX  $#19         Ø1 1Ø    A2 19    load shift counter
LOOP1    ASL  RDR            12    Ø6 3Ø    left shift remainder
         ASL  QUO3           14    Ø6 2F    left shift quotient LSB
LOOP1A   BCS  HIQUO1         16    BØ 28    go to incrementing routine
                                             if carry set
         ASL  QUO2           18    Ø6 2E    left shift quotient mid-byte
         BCS  HIQUO2         1A    BØ 2F    go to incrementing routine
                                             if carry set
         ASL  QUO1           1C    Ø6 2D    left shift quotient MSB
```

```
LOOP2     CLC                  1E    18          clear carry
          ASL   DIVD3          1F    Ø6 2B       left shift dividend LSB
          BCS   HIORD1         21    BØ 2F       go to incrementing routine
                                                    if carry set
          ASL   DIVD2          23    Ø6 2A       left shift dividend mid-byte
          BCS   HIORD2         25    BØ 36       go to incrementing routine
                                                    if carry set
          ASL   DIVD1          27    Ø6 29       left shift dividend MSB
LOOP3     BCS   INCR           29    BØ 39       go to incrementing routine
                                                    if carry set
LOOP4     DEX                  2B    CA          decrement shift counter
          BEQ   FINIS          2C    FØ 3B       jump to end if X=Ø
          SEC                  2E    38          set carry
          LDA   RDR            2F    A5 3Ø       from current remainder
          SBC   DIVIS          31    E5 2C         subtract divisor
          BMI   LOOP1          33    3Ø DD       back to LOOP1 if negative
          STA   RDR            35    85 3Ø       store difference as remainder
          ASL   RDR            37    Ø6 3Ø       left shift remainder
          ASL   QUO3           39    Ø6 2F       left shift quotient LSB
          INC   QUO3           3B    E6 2F       increment quotient LSB
          JMP   LOOP1A         3D    4C 16 Ø1      and go back to LOOP1A
HIQUO1    ASL   QUO2           4Ø    Ø6 2E       left shift quotient mid-byte
          INC   QUO2           42    E6 2E         and increment it
          BCS   HIQUO2         44    BØ Ø5       go to further incrementing
                                                    routine if carry
          ASL   QUO1           46    Ø6 2D       left shift quotient MSB
          JMP   LOOP2          48    4C 1E Ø1      and back to LOOP2 (if C=Ø)
HIQUO2    ASL   QUO1           4B    Ø6 2D       left shift quotient MSB
          INC   QUO1           4D    E6 2D       increment quotient MSB
          JMP   LOOP2          4F    4C 1E Ø1      and back to LOOP2
HIORD1    ASL   DIVD2          52    Ø6 2A       left shift dividend mid-byte
          INC   DIVD2          54    E6 2A       increment dividend mid-byte
          BCS   HIORD2         56    BØ Ø5       go to further incrementing
                                                    routine if carry
          ASL   DIVD1          58    Ø6 29       left shift dividend MSB
          JMP   LOOP3          5A    4C 29 Ø1      and back to LOOP3 (if C=Ø)
HIORD2    ASL   DIVD1          Ø1 5D   Ø6 29      left shift dividend MSB
          INC   DIVD1          5F    E6 29       increment dividend MSB
          JMP   LOOP3          61    4C 29 Ø1      and back to LOOP3
INCR      INC   RDR            64    E6 3Ø       increment remainder
          JMP   LOOP4          66    4C 2B Ø1      and back to LOOP4
FINIS     LSR   RDR            69    46 3Ø       right shift remainder to end
          RTS                  6B    6Ø          return to main program
```

4. SHOWER & TIMER1

Subroutines to generate error message for display on the KIM-1 6-digit LED readout by successive lighting of appropriate segments of the individual digits using a message lookup table.

```
SHOWER requires: subroutines: TIMER1     ØØDE-ØØE9     timing loop for display
                              SHORES     Ø39Ø-Ø3CF     result display for ERROR=Ø1 or Ø2

                 : data arrays: SADD      1741)
                                SBDD      1743)
                                SAD       174Ø)        monitor storage for readout
                                SBD       1742)
                                ERROR     ØØØ2
                                MSGERR    ØØD6-ØØDA
                                MSGNUM    ØØDB-ØØDD
```

```
SHOWER    LDA   $#7F          ØØAØ    A9 7F
          STA   SADD           A2     8D 41 17    set output directional vector A=7F
          LDA   $#1E           A5     A9 1E
          STA   SBDD           A7     8D 43 17    set output directional vector B=1E
DISP2     LDY   $#Ø8           AA     AØ Ø8       set digit selection counter
          LDX   $#Ø5           AC     A2 Ø5       set loop counter
DISP1     STY   SBD            AE     8C 42 17    select digit
          LDA   MSGERR,X       B1     B5 D5       select segments
          STA   SAD            B3     8D 4Ø 17      to be lit (from lookup table)
          JSR   TIMER1         B6     2Ø DE ØØ      and jump to timing loop
          INY                  B9     C8          select next digit
```

```
            INY                 BA    C8
            DEX                 BB    CA              decrement loop counter
            BNE   DISP1         BC    DØ  FØ          if ≠Ø loop again
            LDA   $#12          BE    A9  12
            STA   SBD           CØ    8D  42  17      for sixth digit
            LDX   ERROR         C3    A6  Ø2          set index to error flag
            LDA   MSGNUM,X      C5    B5  DA          and select segments
            STA   SAD           C7    8D  4Ø  17       to be lit (from lookup table)
            JSR   TIMER1        CA    2Ø  DE  ØØ        and jump to timing loop
            LDA   ERROR         CD    A5  Ø2
            CMP   $#Ø3          CF    C9  Ø3          if ERROR=Ø3
            BEQ   DISP2         D1    FØ  D7           loop same display again
            JMP   SHORES        D3    4C  9Ø  Ø3      otherwise jump to show result
```

lookup tables:

```
ØØD6   DØ  DC  DØ  DØ  F9      MSGERR
ØØDB   86  DB  CF              MSGNUM
```

TIMER1 requires: interval timer location 17Ø7

```
TIMER1      LDA   $#FF          ØØDE    A9  FF          set timer for approximately
            STA   17Ø7            EØ    8D  Ø7  17       200 milliseconds per digit
DELAY1      NOP                   E3    EA            do nothing but light segments
            BIT   17Ø7            E4    2C  Ø7  17     time up?
            BPL   DELAY1          E7    1Ø  FA        no? keep lit
            RTS                   E9    6Ø            yes? back to SHOWER for next digit
```

5. SHORES & TIMER2

Subroutines to generate result display on the KIM-1 6-digit LED readout by loading appropriate data into array DISP for display by KIM monitor subroutine SCANDS.

SHORES requires: subroutines: TIMER2 ØØ3DØ-Ø3E5 *timing loop for display*
SHOWER ØØAØ-ØØD5 *error display for ERROR=Ø1 or Ø2*

: data arrays:
ERROR ØØØ2
RESULT ØØØ3-ØØØE
BASE ØØ4A-ØØ4B
NUMBER ØØ4C-ØØ4E
DISP ØØF9-ØØFA *monitor storage for readout:*
ØØF9 INH
ØØFA POINTL
ØØFB POINTH

```
SHORES      LDY   $#Ø1          Ø39Ø    AØ  Ø1          set index for DISP
            LDX   $#Ø3            92    A2  Ø3          set index for NUMBER
LOADN1      LDA   NUMBER,X        94    B5  4B          put NUMBER into DISP
            STA   DISP,Y          96    99  F8  ØØ
            INY                   99    C8              increment DISP index
            DEX                   9A    CA              decrement NUMBER index
            BNE   LOADN1          9B    DØ  F7          loop until DISP is full
            JSR   TIMER2          9D    2Ø  DØ  Ø3       and jump to timing/display loop
            LDA   BASE1           AØ    A5  4A          load BASE1
            STA   POINTH          A2    85  FB           into two highest digits
            LDA   $#BB            A4    A9  BB          load BB
            STA   POINTL          A6    85  FA           into two middle digits
            LDA   BASE2           A8    A5  4B          load BASE2
            STA   INH             AA    85  F9           into two lowest digits
            JSR   TIMER2          AC    2Ø  DØ  Ø3       and jump to timing/display loop
            LDX   $#Ø1            AF    A2  Ø1          set index for RESULT
LOADN3      LDY   $#Ø3            B1    AØ  Ø3          set index for DISP
LOADN2      LDA   RESULT,X        B3    B5  Ø2          put RESULT (3 bytes at a time)
            STA   DISP,Y          B5    99  F8  ØØ       into DISP
            INX                   B8    E8              increment RESULT index
            DEY                   B9    88              decrement DISP index
            BNE   LOADN2          BA    DØ  F7          loop until DISP is full
            TXA                   BC    8A              put RESULT index into accumulator
```

```
          PHA                    BD    48                 and push onto stack
          JSR   TIMER2           BE    2Ø   DØ   Ø3       now jump to timing/display loop
          PLA                    C1    68                pull accumulator from stack
          TAX                    C2    AA                 and put in RESULT index X
          CPX   $#ØD             C3    EØ   ØD           is X > ØC?
          BCC   LOADN3           C5    9Ø   EA           if not, loop back to load DISP
          LDA   ERROR            C7    A5   Ø2           if yes, does ERROR=ØØ?
          CMP   $#ØØ             C9    C9   ØØ
          BEQ   SHORES           CB    FØ   C3           if yes, loop again for whole display
          JMP   SHOWER           CD    4C   AØ   ØØ      otherwise show error


TIMER2 requires: subroutines: SCANDS     1F1F          monitor display subroutine

                 data arrays: CTLP       ØØ49
                 interval timer location 17Ø7

TIMER2    LDA   $#Ø5          Ø3DØ    A9   Ø5
          STA   CTLP            D2    85   49           set loop counter
 DSPN2    LDA   $#FF          Ø3D4    A9   FF           set timer for maximum run
          STA   17Ø7            D6    8D   Ø7   17
 DSPN1    JSR   SCANDS          D9    2Ø   1F   1F      and call display subroutine
          BIT   17Ø7            DC    2C   Ø7   17      time up?
          BPL   DSPN1           DF    1Ø   F8           no? maintain display
          DEC   CTLP            E1    C6   49           decrement loop counter
          BNE   DSPN2           E3    DØ   EF           if ≠Ø, reset timer and maintain display
          RTS                   E5    6Ø                otherwise back to SHORES for next entry
```

LIFE FOR THE KIM-1 AND AN XITEX VIDEO BOARD

Theodore E. Bridge
54 Williamsburg Drive
Springfield, MA 01108

I have been very interested in the game of LIFE ever since I read Martin Gardiner's "Recreational Mathematics" section in the Scientific American - Oct. Nov., 1970. Naturally, I was very much interested in Dr. Frank Covitz' excellent article that appeared on page 5:5 pf the June-July issue of MICRO, 1978.

Just as soon as I got my XITEX video board working on my KIM-1 (16 K on a KIMSI mother board), I attempted to put the Covitz program on my machine. Because the display feature of the XITEX video board is so different from the PET, I thought it was necessary to write a completely new program. I think there may be other KIM-1 users who would like to try my version of this fascinating game.

John Conway invented the game of LIFE. I like to think of it as a simulation of a virus growing on the surface of a POND of DNA. Therefore, I call the work area in which births and deaths are recorded, the POND. I have a routine SHOALL that will display the POND on the screen. I have another routine DISPLY that will add a cell to the screen when a new one is born, and will remove one that is about to die. The POND is updated after each generation in UPDATE. The routine NBRS will record the number of neighbors for a given cell in variable NN. In the pond, zero represents a nonliving cel; (1) represents a living cell; (-1) represents a cell that is about to be born; and (2) represents one that is about to die.

It would take about a second to sweep the entire POND looking for births and deaths, but it takes 1/6 seconds to process a birth or a death. The POND is a matrix 16 x 64. In the routine EDGE, the POND is edged with zeroes to prevent WRAP-AROUND that would destroy symetry in a life form. According to Conway's rules:
1 A new cell is born in an empty cell having 3 neighbors.

2 Any living cell having less than two, or more than three neighbors will die.

3 All deaths and births occur at the same time. A new cell will not be counted as a neighbor until after all cells have oeen processed.

The POND may be relocated on another page by putting the page number at address $2004. Sixty four ($40) bytes must be reserved immediately before and after the POND for edging with zeroes.

START THE PROGRAM AT $2000

The routine PLANT will put a live cell in the center of the screen, and ask for coordinates V , H for other cells, measured from the center. V is the line number (↑ is down and - is up). H is the column number (↑ is right and - is left). Both V and H must be in the range: minus 7 to plus 7. The sign must follow the digit entered, but a space may be substituted for the plus sign. The following entries will establish a blinker in mid screen.

```
ENTER V,H ? 1-,0↑                0
ENTER V,H ? 1↑,0↑                0
ENTER V,H ? /                    0
```

The slash (/) above will terminate the data and start the program.

A generation count is displayed in the upper left corner of the screen. The computer will enter a break if there are no births and no deaths in any generation. To return to the monitor, you will need to insert $1C00 in the IRQ vector. – 17FE 00, and in 17FF 1C.

If your video board uses different commands for positioning the cursor, you will need to change the routine DISPLY. The XITEX board uses the following commands.

```
Key    Hex
       Code
ESC    $1B   invokes coordiante mode
'=     $3D   invokes absolute addressing
"V"    BINARY ROW NUMBER - from top
"H"    BINARY COLUMN NUMBER - from left
           (add $40 if less than $20)
'0     $30   will display a zero
'      $20   will overwrite a cell with
             a space
```

If you have a highspeed video board, you might wish to reform the entire display after each generation with this patch:

```
change Address $204F from EC  to  E9
change Address $2271 from 48  to  60
```

An article by David J. Buckingham in the Dec 1978 issue of BYTE, on page 54 gives a great many life forms that you might like to try with this program.

For practice on inputting data, you might like to try the following life forms given by John Gardner in the Oct.-Nov. 1970 issue of the SCIENTIEIC AMERICAN.

```
000          0+   1+
0            0+   2+
             1+   0
```

Beehive

This fellow lives for four generations and becomes stable in a form called a beehive.

```
000          0+   1+
 0           0+   1-
             1+   0+
```

Traffic Light

After 10 generations, this fellow becomes a blinking traffic light.

```
000          0+   1+
0            0+   2+
 0           1+   0+
             2+   1+
```

Glider

This glider floats up the pond. When he hits the ceiling, he turns into a stable block of four living cells.

```
0000         0+   1+
0   0        0+   2+
 0           0+   3+
             1+   4+
             1+   0+
             2+   0+
             3+   1+
```

Spaceship

This spaceship travels across the pond colliding with the left edge after 10 generations. He then shoots a glider down.

```
0   0        2-   2-
0   0        2-   2+
 000          1-   2-
0   0        1-   2+
0   0        0+   1-
             0+   1+
             1+   2-
             1+   2+
             2+   2-
             2+   2+
```

Spaceman

This life form was first tried by Bob Borg. See figures 1 and 2 for the history of this interesting life form.

If we turn spaceman sideways, he bumps the ceiling after 13 generations losing partial symmetry. He regains symmetry after generation 94. After generation 111, he turns into 2 beehives and four blinkers.

```
   000    000
 0   0    0   0
 00  0    0  00
 0   00  00   0
     00  00
```

```
 0      0    0 0    0
 00   0          0   00
 0    0 0    0 0    0
```

```
     00  00
 0   00  00   0
 00  0    0  00
 0   0    0   0
  000    000
```

Figure 1

This is SPACEMAN after 18 generations. He will soon bump his head on the ceiling just before his feet touch the floor. This will throw him out of symmetry. After generation 33, he will begin to contract to the form displayed in figure 2.

```
  00              00
    0               0
 0  0            0  0
  000             000
```

Figure 2

This is SPACEMAN after 75 generations. This is his minimum size. He will now grow and then later contract again. I have only followed his history through 150 generations.

Johnson lost his microprocessor again

by: Bertha B. Kogut

```
                    CONWAY'S GAME OF LIFE

2000                LIFE    ORG     $2000
2000 4C 2E 20               JMP     START
2003 00             DATA    =       $00
2004 23                     =       $23     FIRST ADDRESS IN POND
                     ALLOW $40 BYTES BEFORE AND
                     AFTER POND FOR WRAP-AROUND.
                     POND IS 1 K  BYTES LONG.
2005 C0                     =       $C0     PON
2006 51                     =       $51
2007 00                     =       $00     LAS
2008 56                     =       $56
2009 00                     =       $00      UL OFFSET
200A 01                     =       $01      UP
200B 02                     =       $02      UR
200C 40                     =       $40      LEFT
200D 42                     =       $42
200E 80                     =       $80      LL
200F 81                     =       $81      DOWN
2010 82                     =       $82      LR
2011                PONDL   *       $001C    FIRST ADDRESS IN POND
2011                PONDH   *       $001D
2011                PON     *       $001E
2011                LAS     *       $0020
2011                OFFSET  *       $0022    DATA WILL BE MOVED HERE
2011                LAST    *       $002A    POINTS TO LAST ADDR. IN POND
2011                ADR     *       $002C    (POINT-POND)=($40*V + H)
2011                V       *       $002E    VERTICAL ORDINATE
2011                H       *       $002F
2011                CNT     *       $0030    COUNT
2011                NN      *       $0031    NUMBER OF NEIGHBORS
2011                LFLAG   *       $0032    LIFE FLAG
2011                SAVY    *       $0033
2011                POINTL  *       $0034
2011                POINTH  *       $0035
2011                POINT   *       $0036
2011                CL      *       $0038
2011                CH      *       $0039

                    KIM ROUTINES

2011 4C 3E 1E       RDTRYT  JMP     $1E3E
2014 84 33          GETCH   STY     SAVY
2016 20 5A 1E               JSR     $1E5A
2019 A4 33                  LDY     SAVY
201B 60                     RTS
201C A9 0D          CRLF    LDAIM   $0D
201E 20 23 20               JSR     OUTCH
2021 A9 0A                  LDAIM   $0A
2023 84 33          OUTCH   STY     SAVY
2025 20 A0 1E               JSR     $1EA0
2028 A4 33                  LDY     SAVY
202A 60                     RTS
```

```
                    BEGIN HERE

202B A0 00      START   LDYIM $00
202D 84 38              STY   GL
202F 84 39              STY   GH
2031 20 53 20           JSR   MOVZ     MOVE DATA TO ZERO PAGE
2034 20 D7 21           JSR   CLEAR
2037 20 2E 21           JSR   PLANT     SEED IN POND
203A 20 A5 21           JSR   SHOALL    OF POND ON TUBE
203D 20 39 22   STAR    JSR   INCG      INCRE. GENER. COUNT
2040 A0 00              LDYIM $00
2042 84 32              STY   LFLAG     ZERO LIVING FLAG
2044 20 11 22           JSR   EDGE       POND WITH ZEROES
2047 20 AF 20           JSR   POST      BIRTHS & DEATHS
204A 20 F1 21           JSR   UPDATE    THE POND
204D A5 32              LDA   LFLAG
204F D0 EC              BNE   STAR      YES. CHECK NEXT GENERATION
2051 00                 BRK
2052 00                 BRK
2053 A2 0D      MOVZ    LDXIM $0D
2055 BD 03 20           LDAAX DATA      GET A DATA WORD
2058 95 1C              STAZX PONDL     PUT IN PAGE ZERO
205A CA                 DEX
205B 10 F8              BPL   MOVZ    +02
205D 18                 CLC
205E A5 1C              LDA   PONDL     POND - $40
2060 69 C0              ADCIM $C0       POND     K       A
2062 85 2A              STA   LAST                R       E
2064 A5 1D              LDA   PONDH             0       R
2066 69 03              ADCIM $03        LAST W       A
2068 85 2B              STA   LAST    +01   LAST +40
206A A5 1D              LDA   PONDH
206C 85 1F              STA   PON     +01
206E C6 1F              DEC   PON     +01
2070 A5 2B              LDA   LAST    +01
2072 85 21              STA   LAS     +01
2074 E6 21              INC   LAS     +01
2076 60                 RTS

                    CALC V & H  FROM ADDRESS IN  ADR

2077 A6 2D      CALCVH  LDX   ADR     +01
2079 A5 2C              LDA   ADR
207B 4C 80 20           JMP   CAL
207E E6 2E              INC   V
2080 38         CAL     SEC
2081 E9 40              SBCIM $40
2083 B0 F9              BCS   CAL     -02
2085 CA                 DEX
2086 10 F6              BPL   CAL     -02
2088 85 2F              STA   H         REMAINDER IN  H
208A 60                 RTS

                    CALC  ADR = POINT - POND

208B 38         CLCADR SEC
```

```
208C A5 34          LDA   POINTL
208E E5 1C          SEC   PONDL
2090 85 2C          STA   ADR
2092 A5 35          LDA   POINTH
2094 E5 1D          SBC   PONDH
2096 85 2D          STA   ADR    +01
2098 60             RTS

                    SET NN = NO. OF NEIGHBORS FOR CELL
                    AT POINT.

2099 20 5F 22  NBRS   JSR   MOV
209C A2 07            LDXIM $07
209E B5 22     NBR    LDAAX OFFSET
20A0 A8               TAY
20A1 B1 36            LDAIY POINT
20A3 F0 04            BEQ   NB       NOT A NEIGHBOR
20A5 30 02            BMI   NB       CONTINUE
20A7 E6 31            INC   NN
20A9 CA        NB     DEX
20AA 10 F2            BPL   NBR
20AC A0 00            LDYIM $00
20AE 60               RTS

                    POST BIRTHS & DEATHS

20AF 20 CC 21  POST   JSR   MOVE     BIRTH = -1
20B2 20 99 20         JSR   NBRS
20B5 A5 31            LDA   NN       ALIVE =+1
20B7 C9 02            CMPIM $02      WILL DIE
20B9 30 13            BMI   DEATH    IF < 2
20BB C9 03            CMPIM $03
20BD F0 1C            BEQ   BIRTH    IF = 3
20BF 10 0D            BPL   DEATH    IF > 3
20C1 20 58 22  POSTA  JSR   INCPT    INCREMENT POINT
20C4 38               SEC
20C5 A5 35            LDA   POINTH
20C7 E5 1D            SBC   PONDH
20C9 C9 04            CMPIM $04
20CB 30 E5            BMI   POST   +03 NOT YET DONE WITH THIS CELL
20CD 60               RTS           NOW WE ARE DONE WITH IT
20CE B1 34     DEATH  LDAIY POINTL
20D0 F0 EF            BEQ   POSTA
20D2 A9 02            LDAIM $02
20D4 91 34            STAIY POINTL
20D6 A9 20            LDAIM $20
20D8 4C E5 20         JMP   BIRTHS
20DB B1 34     BIRTH  LDAIY POINTL
20DD D0 E2            BNE   POSTA
20DF A9 FF            LDAIM $FF
20E1 91 34            STAIY POINTL
20E3 A9 30            LDAIM '0
20E5 20 71 22  BIRTHS JSR   DISPLY
20E8 E6 32            INC   LFLAG
20EA 4C C1 20         JMP   POSTA
```

```
20ED 18          CONVI   CLC
20EE 65 2F               ADC   H
20F0 85 2C               STA   ADR
20F2 90 02               BCC   CONVH   -01
20F4 E6 2D               INC   ADR     +01
20F6 60                  RTS

                  CONVERT H & V TO EQUIV. ADDR.

20F7 A6 2E       CONVH   LDX   V
20F9 A0 00               LDYIM $00
20FB 84 2C               STY   ADR
20FD 84 2D               STY   ADR     +01    CLEAR ADR
20FF CA          CONV    DEX
2100 30 FE               BMI   CONVI
2102 18                  CLC
2103 A9 40               LDAIM $40
2105 65 2C               ADC   ADR
2107 85 2C               STA   ADR
2109 90 F4               BCC   CONV
210B E6 2D               INC   ADR     +01
210D 4C FF 20            JMP   CONV

                  ASK FOR  V,H

2110 20 1C 20    ENTRVH  JSR   CRLF
2113 A2 0E               LDXIM $0E
2115 BD 1F 21            LDAAX ENT
2118 20 23 20            JSR   OUTCH
211B CA                  DEX
211C 10 F7               BPL   ENTRVH +05
211E 60                  RTS
211F 20          ENT     =       '
2120 3F                  =       '?
2121 20                  =       '
2122 48                  =       'H
2123 2C                  =       ',
2124 56                  =       'V
2125 20                  =       '
2126 52                  =       'F
2127 45                  =       'E
2128 54                  =       'T
2129 4E                  =       'N
212A 45                  =       'E

                  PLANT THE SEED

212B A0 00               LDYIM $00
212D 60                  RTS
212E A9 07       PLANT   LDAIM $07
2130 85 2E               STA   V         SET FOR MILSCREEN
2132 A9 1F               LDAIM $1F
2134 85 2F       BACK    STA   H
2136 20 F7 20            JSR   CONVH
2139 18                  CLC
213A A5 2C               LDA   ADR
```

```
213C 65 1C            ADC   PONDL
213E 85 34            STA   POINTL
2140 A5 2D            LDA   ADR     +01
2142 65 1D            ADC   PONDH
2144 85 35            STA   POINTH
2146 A9 01            LDAIM $01
2148 91 34            STAIY POINTL
214A 20 10 21  BASK   JSR   ENTEVH
214D 20 9E 21         JSR   GET
2150 F0 F8            BEQ   BASK
2152 C9 30            CMPIM '0
2154 30 D5            BMI   PLANT   -03
2156 29 07            ANDIM $07
2158 85 2E            STA   V
215A 20 9E 21         JSR   GET
215D F0 EE            BEQ   BASK
215F C9 2D            CMPIM '-
2161 D0 07            BNE   PLAN
2163 38               SEC
2164 A9 00            LDAIM $00
2166 E5 2E            SEC   V
2168 85 2E            STA   V
216A A9 2C     PLAN   LDAIM ',
216C 20 23 20         JSR   OUTCH
216F 18               CLC
2170 A5 2E            LDA   V
2172 69 07            ADCIM $07
2174 85 2E            STA   V
2176 20 9E 21         JSR   GET
2179 F0 CF            BEQ   BASK
217B C9 30            CMPIM '0
217D 30 AC            BMI   PLANT   -03
217F 29 07            ANDIM $07
2181 85 2F            STA   H
2183 20 9E 21         JSR   GET
2186 F0 C2            BEQ   BASK
2188 C9 2D            CMPIM '-
218A D0 07            BNE   PLANTE
218C 38               SEC
218D A9 00            LDAIM $00
218F E5 2F            SEC   H
2191 85 2F            STA   H
2193 A5 2F     PLANTE LDA   H
2195 18               CLC
2196 69 1F            ADCIM $1F       MEASURE TO CENTER
2198 4C 34 21         JMP   BACK

                  GET A COORDINATE

219B 20 14 20  GET    JSR   GETCH
219E C9 38            CMPIM '8
21A0 30 02            BMI   BAD
21A2 A9 00            LDAIM $00
21A4 60        BAD    RTS

                  DISPLAY ALL OF FONT
```

```
21A5 20 CC 21   SHOALL JSR   MOVE
21A8 A9 0F              LDAIM $0F
21AA 85 2E              STA   V
21AC A9 3F      SHOAL   LDAIM $3F
21AE 85 2F              STA   H
21B0 20 1C 20           JSR   CRLF
21B3 B1 34      SHOA    LDAIY POINTL
21B5 F0 04              BEQ   SHO
21B7 A9 30              LDAIM '0
21B9 10 02              BPL   SHO     +02
21BB A9 20      SHO     LDAIM $20
21BD 20 23 20           JSR   OUTCH
21C0 20 58 22           JSR   INCPT
21C3 C6 2F              DEC   H
21C5 10 EC              BPL   SHOA
21C7 C6 2E              DEC   V
21C9 10 E1              BPL   SHOAL
21CB 60                 RTS

                MOVE POND TO POINT

21CC A5 1C      MOVE    LDA   PONDL
21CE 85 34              STA   POINTL
21D0 A5 1D              LDA   PONDH
21D2 85 35              STA   POINTH
21D4 A0 00              LDYIM $00
21D6 60                 RTS

                 CLEAR POND

21D7 20 CC 21   CLEAR   JSR   MOVE
21DA A9 0F              LDAIM $0F
21DC 85 30              STA   CNT
21DE A2 3F              LDXIM $3F
21E0 98                 TYA
21E1 91 34      CLEA    STAIY POINTL
21E3 20 58 22           JSR   INCPT
21E6 CA                 DEX
21E7 10 F8              BPL   CLEA
21E9 C6 30              DEC   CNT
21EB 10 F1              BPL   CLEA    -03
21ED 20 CC 21           JSR   MOVE
21F0 60                 RTS

            BURY THE DEAD AND RAISE THE CHILDREN

21F1 20 CC 21   UPDATE  JSR   MOVE
21F4 B1 34              LDAIY POINTL
21F6 30 08              BMI   POSTIT -02
21F8 C9 02              CMPIM $02
21FA 30 08              BMI   POSTIT +02
21FC A9 00              LDAIM $00
21FE F0 02              BEQ   POSTIT
2200 A9 01              LDAIM $01
2202 91 34      POSTIT  STAIY POINTL
```

```
2204 20 58 22          JSR   INCPT
2207 A5 35             LDA   POINTH
2209 C5 21             CMP   LAS     +01
220B 30 E7             BMI   UPDATE +03
220D 20 CC 21          JSR   MOVE
2210 60                RTS

                  EDGE POND WITH ZEROES
                  TO PREVENT WRAP-AROUND

2211 20 CC 21  EDGE    JSR   MOVE
2214 A0 3F             LDYIM $3F
2216 A9 00             LDAIM $00
2218 91 1E             STAIY PON
221A 91 20             STAIY LAS
221C 88                DEY
221D 10 F9             BPL   EDGE    +07
221F A0 00             LDYIM $00
2221 A5 34       WRA   LDA   POINTL
2223 18                CLC
2224 69 40             ADCIM $40
2226 85 34             STA   POINTL
2228 A9 00             LDAIM $00
222A 65 35             ADC   POINTH
222C 85 35             STA   POINTH
222E C5 21             CMP   LAS     +01
2230 B0 DE             BCS   EDGE    -01
2232 A9 00             LDAIM $00
2234 91 34             STAIY POINTL
2236 4C 21 22          JMP   WRA

                  INCREMENT AND DISPLAY
                  THE GENERATION COUNT

2239 18          INCG  CLC
223A F8                SED
223B A9 01             LDAIM $01
223D 65 38             ADC   GL
223F 85 38             STA   GL
2241 A9 00             LDAIM $00
2243 65 39             ADC   GH
2245 85 39             STA   GH
2247 D8                CLD
2248 A9 04       NCG   LDAIM $04
224A 20 23 20          JSR   OUTCH
224D A5 39             LDA   GH
224F 20 11 20          JSR   PRTBYT
2252 A5 38             LDA   GL
2254 20 11 20          JSR   PRTBYT
2257 60                RTS
2258 E6 34       INCPT INC   POINTL
225A D0 02             BNE   INCPT   +06
225C E6 35             INC   POINTH
225E 60                RTS
225F 38          MOV   SEC
2260 A5 34             LDA   POINTL
```

```
2262 E9 41            SBCIM $41
2264 85 36            STA   POINT
2266 A5 35            LDA   POINTH
2268 E9 00            SBCIM $00
226A 85 37            STA   POINT  +01
226C A0 00            LDYIM $00
226E 84 31            STY   NN
2270 60               RTS

                 DISPLAY  THE CHARACTER IN THE ACC.
                 AT THE -- POINT -- ADDRESS ON TUBE

2271 48         DISPLY PHA            SAVE ACC
2272 20 8E 20          JSR   CLCADR
2275 84 2E             STY   V
2277 20 77 20          JSR   CALCVH    CALC V,H
227A A9 1E             LDAIM $1E       PRINT ESCAPE
227C 20 23 20          JSR   OUTCH     TO MOVE CURSOR
227F A9 3D             LDAIM '=        AES ADDRESS
2281 20 23 20          JSR   OUTCH
2284 A5 2E             LDA   V
2286 09 40             ORAIM $40       ADJSUT  V
2288 20 23 20          JSR   OUTCH
228B A5 2F             LDA   H         ADJUST  H
228D C9 20             CMPIM $20
228F 10 02             BPL   DISP
2291 09 40             ORAIM $40
2293 20 23 20  DISP    JSR   OUTCH
2296 68                PLA             GET ACC
2297 20 23 20          JSR   OUTCH     PRINT IT
229A 60                RTS
```

SYMBOL TABLE

		ADR	002C	BACK	2134	BAD	21A4
BASK	214A	BIRTH	20DB	BIRTHS	20E5	CALCVH	2077
CAL	2080	CLCADR	208E	CLEA	21E1	CLEAR	21D7
CNT	0030	CONV	20FF	CONVH	20F7	CONVI	20ED
CRLF	201C	DATA	2003	DEATH	20CB	DISP	2293
DISPLY	2271	EDGE	2211	ENTRVH	2110	ENT	211F
GETCH	2014	GET	219E	GH	0039	GL	0038
H	002F	INCC	2239	INCRT	2258	LAST	002A
LAS	0020	LFLAG	0032	LIFE	2000	MOVE	21CC
MOVZ	2053	MOV	225F	NE	20A9	NEFS	2099
NEP	209F	NCC	2248	NN	0031	OFFSET	0022
OUTCH	2023	PLAN	216A	PLANT	212E	PLANTE	2193
POINT	0036	POINTH	0035	POINTL	0034	PONDH	001D
PONDL	001C	PON	001E	POST	20AF	POSTA	20C1
POSTIT	2202	PRTBYT	2011	SAVY	0033	SHOA	21D3
SHOAL	21AC	SHOALL	21A5	SHO	21BE	STAR	203D
START	202F	UPDATE	21F1	V	002E	WFA	2221

EKIM OR MAXI-KIM
Extended Keyboard Input Monitor

Andrew V.W. Sensicle
155 Valois Bay Ave.
Pointe Claire, Montreal
Quebec, Canada H9R 4B8

Although KIM-1's ROM contains useful features like the tape and TTY input-output routines, when it comes to inputting data or coding via the key pad, KIM's resident monitor leaves much to be desired, for example the avoidance of repetitive pushing of the "↑" between each entry or the ability to look back a few bytes without going into address mode. I would like to thank Jim Butterfield for his excellent BROWSE and BRANCH PROGRAMS which I put together in Page 1 and have used religiously since I got started in this game in mid '78.

However, these have their limitations and I have frequently found the need for a little more sophistication, not to mention the space they occupy in Page 1. Anyway the thing which irritated me most was the need to re-enter a long listing merely in order to open up a few spaces for additional instructions. The process of tidying up a finished program, entailing closing up unwanted spaces and the associated readdressing was also very time consuming.

I thus decided to try to write an extended monitor which would be compact enough to fit in Page 17 and yet provide the functions I needed. After much condensing and compressing I ended up with a program 6 bytes longer than the "legal" Page 17 RAM, but by stealing a little from KIM it fits nicely. KIM doesn't seem to mind. As long as you don't use the tape or TTY routines, he leaves you alone.

The **NMI** vector is loaded with the start address (1780) so that the **ST** key can be used to access the monitor at any open cell address. Before pressing **ST** or after exitting via **RS** the resident monitor is used as a normal in the **AD** mode. The **ST** key gives you 6 other modes of operation or functions.

1. **STAND BY MODE [ST]:** This starts the program which then sits looking at the open cell address and its contents, ie. nothing seems to happen. However, any HEX key is stored at the open cell address which each second key stroke increments the address.

2. **INCREMENT** [↑]: Big deal! This works just like normal.

3. **DECREMENT [PC]:** This steps the address points backwards exactly the reverse of "↑".

4. **OPEN UP MODE [AD]:** Each depression of this key causes one full page of bytes (FF) to be moved one place up starting at the open cell address.

5. **CLOSE UP MODE [DA]:** Each depression of this key causes one full page of bytes to be moved one place back to overwrite the open cell contents. Having made an "open up" or close up move of one or more steps you will, of course, have to fix up all affected addresses. This is not as onerous as it sounds if you use the sixth mode.

6. **BRANCH MODE [GO]:** When a branch instruction is encountered while entering a new program or fixing up an old one, all you need do is press "GO" followed by the actual destinction address (low order only). The monitor will calculate the relative address, store it in the open cell and step on to the next cell all in the twinkling of an eye. The user is, as usual, responsible for ensuring that the branch does not exceed the normal half page range.

I hope that this little program will be as useful to others as it is and has been to me.

```
                        ORG    $1780

MODE    *       $00FF
TEMPX   *       $00FD
LAST    *       $00F3
INL     *       $00F8
POINTL  *       $00FA
POINTH  *       $00FB

SCAND   *       $1F19
GETKEY  *       $1F6A
UPDATE  *       $1FBB
INCPT   *       $1F63

1780 D8         START   CLD
1781 A2 01              LDXIM $01      INITIATE MODE AND
1783 86 FF              STX   MODE     COUNTER
1785 86 FD              STX   TEMPX
```

```
1787 20 19 1F  GETK    JSR    SCAND   LIGHT DISPLAY
178A 20 6A 1F          JSR    GETKEY  CHECK KEYS
178D C5 F3             CMP    LAST
178F F0 F6             BEQ    GETK
1791 85 F3             STA    LAST    NEW KEY
1793 C9 13             CMPIM  $13     GO ?
1795 D0 02             BNE    SKIP
1797 C6 FF             DEC    MODE    PUT IN BRANCH MODE
1799 C9 12      SKIP   CMPIM  $12     + ?
179B F0 4A             BEQ    INCPNT
179D C9 14             CMPIM  $14     PC ?
179F F0 22             BEQ    DECPNT
17A1 C9 11             CMPIM  $11     DA ?
17A3 F0 11             BEQ    CLOSUP
17A5 C9 10             CMPIM  $10     AD ?
17A7 D0 26             BNE    INDATA

17A9 A0 FF      OPENUP LDYIM  $FF     LOAD 255(10)
17AB 88         OPENX  DEY
17AC B1 FA             LDAIY  POINTL  LOAD AND STORE
17AE C8                INY            ONE CELL HIGHER
17AF 91 FA             STAIY  POINTL
17B1 88                DEY
17B2 D0 F7             BNE    OPENX   NEXT
17B4 F0 CA             BEQ    START

17B6 A0 01      CLOSUP LDYIM  $01
17B8 B1 FA      CLOSY  LDAIY  POINTL  LOAD OPEN CELL
17BA 88                DEY            PLUS 1
17BB 91 FA             STAIY  POINTL  STORE IN OPEN CELL
17BD C8                INY            THEN UP
17BE C8                INY            UNTIL
17BF D0 F7             BNE    CLOSY
17C1 F0 BD             BEQ    START   CONE 255 (10)

17C3 C6 FA      DECPNT DEC    POINTL
17C5 A5 FA             LDA    POINTL
17C7 C9 FF             CMPIM  $FF     PAGE CHANGE?
17C9 D0 B5             BNE    START   NO
17CB C6 FB             DEC    POINTH  YES, THEN DEC POINTH
17CD 10 B1             BPL    START   AS WELL

17CF C9 10      INDATA CMPIM  $10
17D1 B0 B4             BCS    GETK    FALSE START ACTUALLY NO KEY
17D3 20 BB 1F          JSR    UPDATE  ROL 4 BITS FROM A TO INL
17D6 A5 F8             LDA    INL
17D8 91 FA             STAIY  POINTL
17DA C6 FD             DEC    TEMPX
17DC F0 A9             BEQ    GETK    ONE MORE KEY
17DE A4 FF             LDY    MODE    IN BRANCH MODE?
17E0 D0 05             BNE    INCPNT  NO
17E2 18                CLC
17E3 E5 FA             SBC    POINTL  CALC RELATIVE ADDRESS
17E5 91 FA             STAIY  POINTL  STORA IT IN OPEN CELL
17E7 20 63 1F  INCPNT  JSR    INCPT   NEW CELL
17EA 4C 80 17          JMP    START   RETURN
```

CORRECTED KIM FORMAT LOADER FOR SYM-1

Nicholas J. Vrtis
5863 Pinetree S.E.
Kentwood, MI 49508

My cassette is an old model GE, and it won't quite hack the high speed tape format of the SYM-1, so I have probably used the KIM format option more than most SYM owners. In the process, I have found a bug in the SYM monitor tape load routine. Synertek knows about the problem, but didn't have a nice fix when I called, so I worked up the attached program.

The problem with the monitor routines is that they will not load a slash (hex 2F) from a KIM format tape. The slash is used to indicate that the data is done, and the checksum follows. The monitor routines don't check for the slash until after the KIM characters have been read and combined. The error you get is a checksum error (ER CC).

Most of the code for this program has been copied from the SYM monitor routines, except these work. The basic logic change is that when a slash is read as a single KIM byte, it is treated as a non-hex character. The non-hex routine checks for the slash instead of after every character. If it is a slash, it goes to the checksum check routine.

This routine is not as fancy as the monitor routines, but it sure beats re-keying a couple K bytes of program. It has turned out to be convenient to have this program available even for loading programs without the slash. By changing the branch after the compare for the slash to a branch back to LOADT7 it will ignore errors. Sometimes this will load a bad tape with only minor errors. Other times the program gets out of sync and loads garbage. It is worth the try for a tape you have spent a lot of time on.

One final comment about cassettes. If you have the remote control connected, putting a hex CC into location AOOC will turn the cassette motor back on. It is easier than yanking the remote plug.

```
                FIXED SYM-1 KIM FORMAT LOADER

                NICHOLAS J. VRTIS
                MARCH 1979

                STRIPPED DOWN VERSIONS OF L1 COMMAND.
                WILL LOAD A 2F WHICH CAUSES SYM-1 TROUBLE.
                ONLY FOR KIM FORMAT TAPES.
                ID SHOULD BE PUT INTO LOCATION 0000.

0080            CHAR    *       $00FC   CHAR ASSEMBLY & DISASSEMBLY
0080            MODE    *       $00FD
0080            BUFADL  *       $00FE   CURRENT CHAR INDIRECT ADDRESS
0080            BUFADH  *       $00FF

                SYM-1 REFERENCES

0080            DDRIN   *       $A002
0080            VIAACR  *       $A00B
0080            LATCHL  *       $A004

0080            ACCESS  *       $8BA6
0080            SLASH   *       $8D3C   SLASH IN SYM MONITOR
0080            LOADTX  *       $8D4F
0080            NHERR   *       $8D69
0080            SYNC    *       $8D82
0080            START   *       $8DB6
0080            RDBYTX  *       $8E28
0080            PACKT   *       $8E3E
0080            RDCHT   *       $8E61
0080            CHKT    *       $8E78
```

```
0000                    ORG    $0000

0000 00         ID      =      $00     RESERVED FOR PROGRAM ID

0001 20 A6 8B  LOADT    JSR    ACCESS UN-PROTECT SYSTEM RAM
0004 A0 00              LDYIM  $00    SET KIM MODE
0006 20 B6 8D           JSR    START  INITIALIZE
0009 AD 02 A0           LDA    DDRIN
000C 29 BF              ANDIM  $BF    BIT 6 = 0 INPUT IS PB6
000E 8D 02 A0           STA    DDRIN
0011 A9 00              LDAIM  $00
0013 8D 0B A0           STA    VIAACR
0016 A9 AE              LDAIM  $AE    SET UP CLOCK FOR GETTR (KIM)
0018 8D 04 A0           STA    LATCHL STORE GETTR VALUE IN LO LATCH
001B 20 82 8D  LOADTA   JSR    SYNC   GET IN SYNC
001E 20 61 8E  LOADTB   JSR    RDCHT
0021 C9 2A              CMPIM  '*     START OF DATA ?
0023 F0 06              BEQ    LOADTC
0025 C9 16              CMPIM  $16    NO - SYNC CHARACTER?
0027 D0 F2              BNE    LOADTA IF NOT, RESTART SYNC SEARCH
0029 F0 F3              BEQ    LOADTB IF YES, KEEP LOOKINT FOR THE *

002B A9 00     LOADTC   LDAIM  $00    CLEAR "NOT IN SYNC BIT"
002D 85 FD              STA    MODE


002F 20 28 8E           JSR    RDBYTX READ ID BYTE

                CHANGE THE FOLLOWING IF ID LOCATION IS
                NOT HEX 0000

0032 C5 00              CMP    ID     COMPARE WITH REQUESTED ID
0034 F0 02              BEQ    LOADTD GO LOAD IF EQUAL
0036 D0 E3              BNE    LOADTA UNCONDITIONAL - RESTART SEARCH

0038 20 28 8E  LOADTD   JSR    RDBYTX GET SAL FROM TAPE
003B 20 78 8E           JSR    CHKT
003E 85 FE              STA    BUFADL PUT IN BUF START LOW
0040 20 28 8E           JSR    RDBYTX SAME FOR SAH
0043 20 78 8E           JSR    CHKT
0046 85 FF              STA    BUFADH

                THE FOLLOWING JSR RDBYT IS THE ONLY
                INSTRUCTION THAT WOULD HAVE TO CHANGE
                TO RE-LOCATE THIS PROGRAM

0048 20 67 00  LOADTE   JSR    RDBYT  GET A BYTE INPUT
004B B0 0F              BCS    XNHERR BRANCH IF NON-HEX
004D 20 78 8E           JSR    CHKT   INCLUDE IN CHECKSUM
0050 A0 00              LDYIM  $00    STORE BYTE
0052 91 FE              STAIY  BUFADL
0054 E6 FE              INC    BUFADL BUMP BUFFER ADDRESS
0056 D0 F0              BNE    LOADTE BRANCH IF NO CARRY
0058 E6 FF              INC    BUFADH ELSE NEED TO UPDATE HIGH ORDER
005A D0 EC              BNE    LOADTE UNCONDITIONAL
```

```
005C CD 3C 8D  XNHERR CMP   SLASH  "/" IN SYM MONITOR
005F D0 03            BNE   YNHERR WAS IT REALLY AN ERROR
0061 4C 4F 8D         JMP   LOADTX NOW LET HIM HANDLE CHECKSUM
0064 4C 69 8D  YNHERR JMP   NHERR  LET MONITOR DO THIS ALSO

0067 20 61 8E  RDBYT  JSR   RDCHT  READ ONE HALF
006A CD 3C 8D         CMP   SLASH  SEE IF A SLASH
006D D0 02            BNE   RDBYTA BRANCH IF NOT
006F 38               SEC         SET CARRY AS NON-HEX
0070 60               RTS         AND RETURN

0071 20 3E 8E  RDBYTA JSR   PACKT  SEE IF GOOD CHARACTER
0074 90 01            BCC   RDBYTB BRANCH AROUND RETURN IF HEX
0076 60               RTS

0077 AA        RDBYTB TAX         SAVE MSD
0078 20 61 8E         JSR   RDCHT  GET NEXT HALF CHARACTER
007B 86 FC            STX   CHAR   SAVE IT HERE
007D 4C 3E 8E         JMP   PACKT  CHECK FOR HEX & RETURN
```

STORAGE SCOPE REVISITED

Joseph L. Powlette
Donald C. Jeffery
Hall of Science
Moravian College
Bethlehem, PA 18018

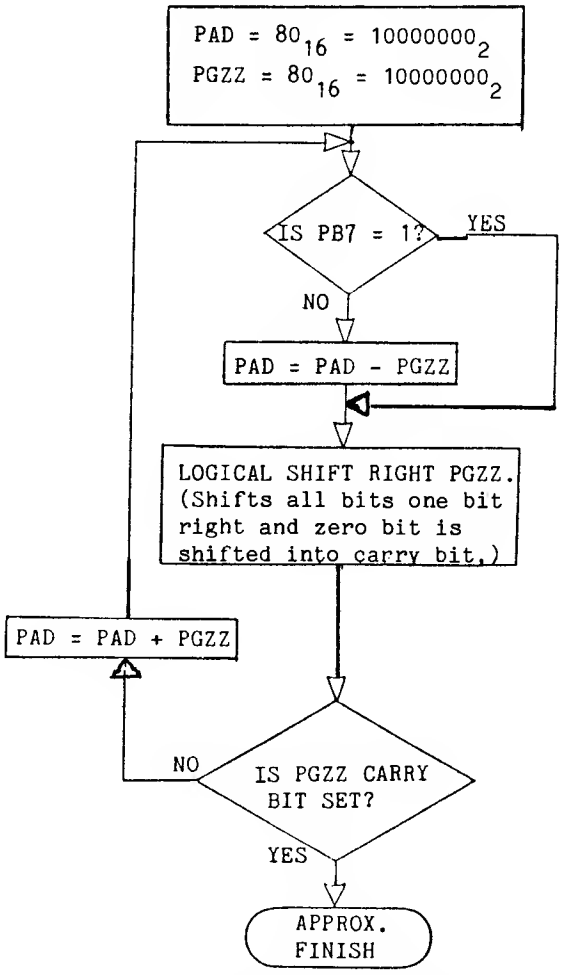
Marvin DeJong has written an excellent article (MICRO, No.2, pp.11-15, Dec 77-Jan 78) which serves to transform an ordinary oscilloscope into a storage scope. We have constructed several units for use in our laboratory and found them to be very useful. However, we would like to suggest a simple hardware change which will improve the quality of the circuits performance. Figure 1 is a photograph of the storage scope response to a triangular wave (14Hz and voltage offset) using DeJong's circuit. The cause of the irregularities seen in this figure was traced to the second OP-AMP which is used as a comparator. The slew rate of the CA3140 is not high enough to adequately accommodate the successive approximation software routine. Figure 2 shows the collection of data for the same wave with the second OP-AMP changed to a 531 high slew rate OP-AMP. The 531, which is readily available, has the same pin-out (in the TO-5 package) as the CA3140 but pin 4 must be connected to -15 volts rather than ground potential. Also, do not use a frequency compensation capacitor with the 531 since this will only decrease the slew rate of this OP-AMP in the comparator configuration. The 531 is not a FET input type and does not have the high input impedance (1.5 T) of the CA3140. If such a high impedance is desirable, one can use a CA3140 in the following configuration preceding the 531 non-inverting voltage input.

One should also note that:

1. There is a 7 bit version of the 1408 DAC. Specify 1408L8 for the 8 bit converter.
2. +5 volts should be connected to pin 13 of the 1408 (see MICRO, No. 6, p. 4, Aug-Sept, 1978)
3. The flow chart for the successive approximation routine is not correct.

DeJong is to be commended for this storage scope application. In fact, the performance of the program (with the above hardware change) approaches that of commercial units.

Flow Chart for
Successive Approximation
Analog to Digital Conversion



Correction to Successive Approximation -
Micro, No.2, P. 13 Dec. 77 - Jan. 78

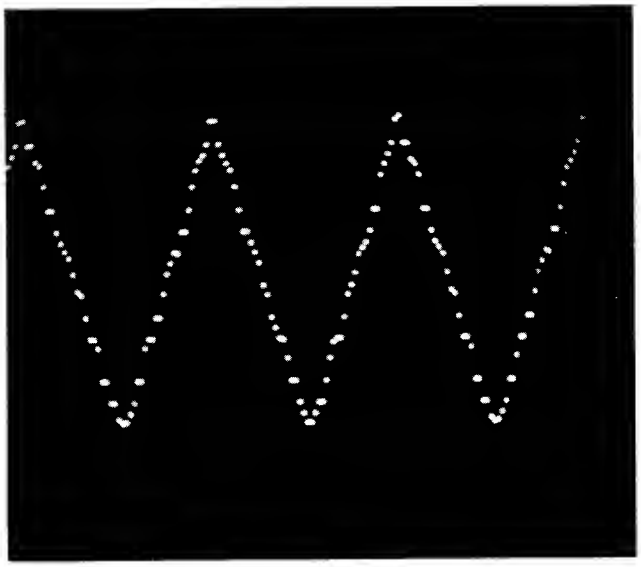


Figure 1

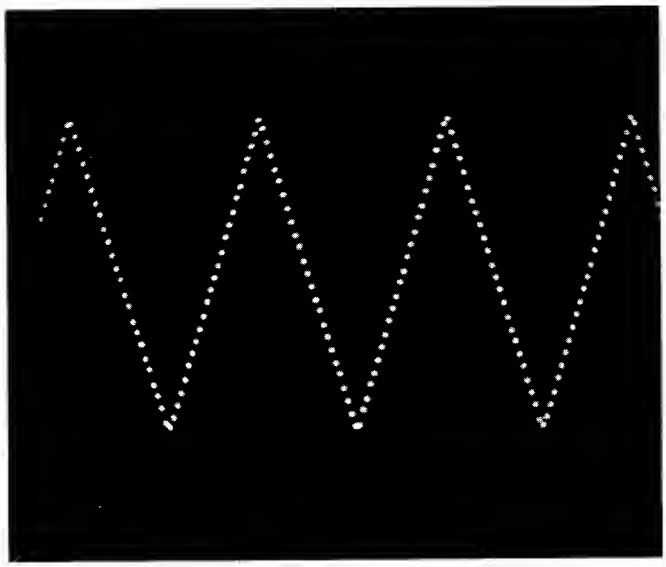
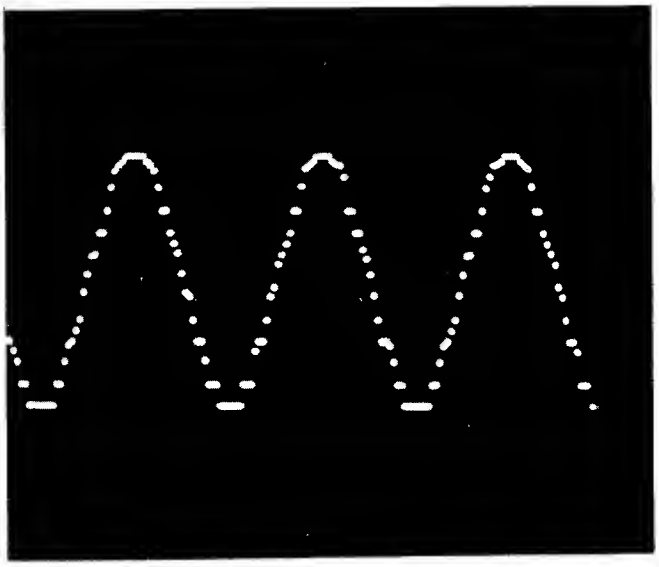
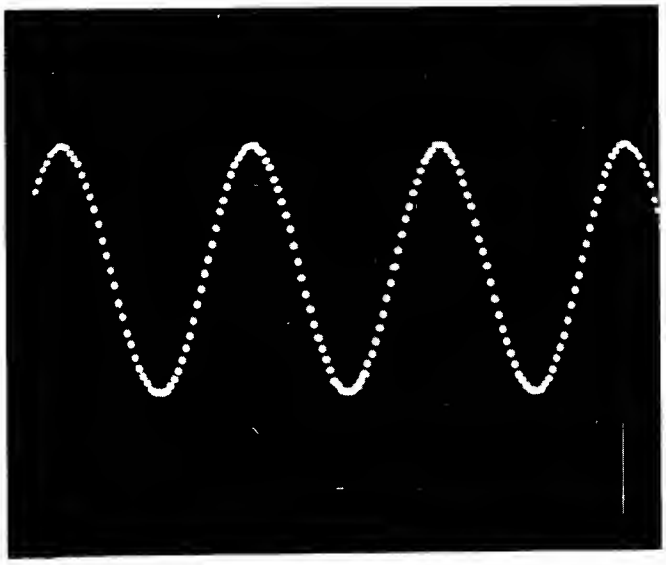


Figure 2



14 Hz Sine Wave
(Voltage Offset)

De Jong's Circuit



14 Hz Sine Wave
(Voltage Offset)

Modified Circuit

APPLE

BREAKER: AN APPLE II DEBUGGING AID

Rick Auricchio
59 Plymouth Ave.
Maplewood, NJ 07040

When debugging an Assembly-language program, one of the easiest tools the programmer can use is the Breakpoint. In its most basic form, the Breakpoint consists of a hardware feature which stops the CPU upon accessing a certain address; a "deluxe" version might even use the Read/Write and Sync (instruction fetch) lines to allow stopping on a particular instruction, the loading of a byte, or the storing of a byte in memory. Since software is often easier to create than hardware (and cheaper for some of us!), a better method might be to implement the Breakpoint with software, making use of the BRK opcode of the 6502 CPU.

A Breakpoint, in practice, is simply a BRK opcode inserted over an existing program instruction. When the user program's execution hits the BRK, a trap to the Monitor (via the IRQ vector $FFFE/FFFF) will occur. In the APPLE, the Monitor saves the user program's status and registers, then prints the registers and returns control to the keyboard. The difficult part, however, comes when we wish to resume execution of the program: the BRK must be removed and the original instruction replaced, and the registers must be restored prior to continuing execution. If we merely replace the original opcode, however, the BRK will not be there should the program run through that address again.

The answer to this problem is BREAKER: a software routine to manage Breakpoints. What the debugger does is quite simple: it manages the insertion and removal of breakpoints, and it correctly resumes a user program after hitting a breakpoint. The original instruction will be executed automatically when the program is resumed!

Is it Magic?

No, it's not magic, but a way of having the computer remember where the breakpoints are! If the debugger knows where the breakpoints are, then it should also know what the original instruction was. Armed with that information, managing the breakpoints is easy. Here's how the debugger works:

During initialization, BREAKER is "hooked-in" to the APPLE monitor via the Control-Y user command exit, and via the COUT user exit. The control-Y exit is used to process debugger commands, and the COUT exit is used to "steal control" from the Monitor when a BRK occurs.

Breakpoint information is kept in tables: the LOCTAB is a table of 2-byte addresses--it contains the address at which a breakpoint has been placed. The ADTAB is a table of 1-byte low-order address bytes; it is used to locate a Break Table Entry (BTE for short). The BTE is 12 bytes long (only the first 9 are used, but 12 is a reasonably round number) and it contains the following items:

* Original user-program instruction
* JMP back to user-program
* JMP back for relative branch targets

When adding a breakpoint, we must build the BTE correctly, and place the user-program break address into the LOCTAB. There are eight (8) breakpoints allowed, so that we have a 16-byte LOCTAB, 8-byte ADTAB, and 96 bytes of BTE's.

As the breakpoint is added, the original instruction is copied to the first 3 bytes of the BTE, and it is "padded" with NOP instructions ($EA) in case it is a 1 or 2-byte instruction. A BRK opcode ($00) is placed into the user program in place of the original instruction's opcode (other instruction bytes are not altered). The next 3 bytes of the BTE will contain a JMP instruction back to the next user-program instruction.

If the original instruction was a Relative Branch, one more thing must be considered: if we remove the relative branch to the BTE, how will it branch correctly? This problem is solved by installing another JMP instruction into the BTE for a relative branch--back to the Target of the branch, which is computed by adding the original PC of the branch, +2, +offset. This Absolute address will be placed into the JMP at bytes 7-9 of the BTE. The offset which was copied from the original instruction will be changed to $04 so that it will now branch to that second JMP instruction within the BTE; the JMP will get us to the intended target of the original Relative Branch.

A call to the routine "INSDS2" in the Monitor returns the length and type of an instruction for the "add" function. The opcode is supplied in the AC, and LENGTH & FORMAT are set appropriately by the routine.

Removal of a breakpoint involves simply restoring the original opcode, and clearing the LOCTAB to free this breakpoint's BTE.

Displaying of breakpoints prints the user-program address of a breakpoint, followed by the address of the BTE associated with the breakpoint (the BTE address is useful--its importance will be described later).

When the breakpoint is executed, a BRK occurs and the APPLE Monitor gets control. The monitor will "beep" and print the user program's registers. During printing of the registers, BREAKER will take control via the COUT exit. (Remember, we get control on every character printed - but it's only important when the registers are being printed. That's when we're at a breakpoint). While it has control, BREAKER will grab the user-program's PC and save it (we must subtract 2 because of the action of the BRK instruction). If no breakpoint exists at this PC (we scan LOCTAB), then the Mointor is continued. If a breakpoint does exist here, then the BTE address is set as the "continue PC". In other words, when we continue the user program after the break, we will go to the BTE; the original instruction will now be executed, and we will branch back to the rest of the user program.

Using BREAKER

The first thing to do is to load BREAKER into high memory. It must then be initialized via entry at the start address. This sets up the exits from the Monitor. After a Reset, you must re-initialize via "YcI" to set up the COUT exit

again. Upon entry at the start address, all breakpoints are cleared; after "YcI", they remain in effect.

To add a breakpoint, type: aaaaYcA . (Yc is control-Y). This will add a breakpoint at address 'aaaa' in the user program. A 'beep' indicates an error; you already have a breakpoint at that address. To remove a breakpoint, type: aaaaYcR. This will remove the breakpoint at address 'aaaa' and restore the original opcode. A 'beep' means that there was none there to start with.

Run your user-program via the Monitor's "G" command. Upon hitting a breakpoint, you will get the registers printed, and control will go back to the monitor as it does normally. At this point, all regular Monitor commands are valid, including "YcA", "YcR", and "YcD" for BREAKER.

To continue execution (after looking at stuff maybe modifying some things), type: YcG . This instructs BREAKER to resume execution at the BTE (to execute the original instruction), then to transfer control back to the user program. Do not resume via Monitor "G" command--it won't work properly, since the monitor knows nothing of breakpoints. To display all breakpoints, type: YcD. This will give a display of up to 8 breakpoints, with the address of the associated BTE for each one.

Caveats

Some care must be taken when using BREAKER to debug a program. First, there is the case of BREAKER not being initialized when you run the user program. This isn't a problem when you start, because you'll not be able to use the Yc commands. But if you should hit Reset during testing, you must re-activate via "YcI", otherwise BREAKER won't get control on a breakpoint. If you try a YcG, unpredictable things will happen. If you know that you hit a breakpoint while BREAKER was not active, you can recover. Simply do a "YcI", and then display the breakpoints (YcD). Resume the user-program by issuing a Monitor "G" command to the BTE for the breakpoint that was hit (since BREAKER wasn't around when you hit the breakpoint, you have to manually resume execution at the BTE). Now all is back to normal. You can tell if BREAKER is active by displaying locations $38 and $39. If not active, they will contain $F0 FD.

It's also important to note that any user program which makes use of either the Control-Y or COUT exits can't be debugged with BREAKER. Once these exits are changed, BREAKER won't get control when it's supposed to.

BREAKER DEBUGGER: Routines to Handle up to 8 Breakpoints, for use in Debugging of User Code.

```
                        **** APPLE-2 MONITOR EQUATES
                        *
002F                    FORMAT  EQU     X'2F'           INSTPUCTION FORMAT
002F                    LENGTH  EQU     X'2F'           INSTRUCTION LENGTH
003C                    A1L     EQU     X'3C'           WORK AREA
003D                    A1H     EQU     X'3D'
003F                    A2L     EQU     X'3E'
003F                    A2H     EQU     X'3F'
0040                    A3L     EQU     X'40'
0041                    A3H     EQU     X'41'
                        *
0036                    CSWL    EQU     X'36'           COUT SWITCH WOPD
0037                    CSWH    EQU     X'37'
                        *
F88E                    INSDS2  EQU     X'F88E'         DISASSEMBLEP
F940                    PRNTYX  EQU     X'F940'         PRINT Y/X PEGS IN HEX
FDDA                    PPBYTE  EQU     X'FDDA'         PRINT AC IN HEX
FDED                    COUT    EQU     X'FDED'         CHAP OUT
FF65                    PESET   EQU     X'FF65'         MONITOR PESET
FF69                    MON     EQU     X'FF69'         MONITOR ENTRY
                        *
                        * CHANGE 'LOWPAGE' TO LOCATE
                        *  ELSEWHEPE IN MEMOPY. IT IS
                        *  NOW SET FOP A 32K SYSTEM.
                        *
   0000007D             LOWPAGE EQU     X'7D'           3 PGS BEFORE END MEMORY
7D00                            ORG     LOWPAGE**8      ORG OUT TO MEMORY TOP
7D00    4C 36 7F        INIT    JMP     INITX           =>INITIALIZATION FNTRY
                        *
                        * --- DATA AREAS --- *
                        *
7D03    00              FW1     DC      0               'FINDPC' WOPK BYTE 1
7D04    00              FW2     DC      0               'FINDPC' WOPK BYTF 2
7D05    00              PCL     DC      0               'GO' PC LO
7D06    00              PCH     DC      0               'GO' PC HI
                        *
                        ** SKELETON BPEAK-TABLE ENTRY (BTE) **
                        *
7D07    00              SKEL    DC      0               SKELETON BTE
7D08    EA                      NOP                     NOPS FOP PADDING
7D09    EA                      NOP
7D0A    4C 00 00                JMP     0               JUMP BACK INLINE
7D0D    4C                      DC      X'4C'           JUMP OPCODE FOP BPANCHES
                        *
```

```
                        *
                        * -- LO ADDRESS OF BTE'S KEPT IN ADTAB -- *
                        *
7D0E       26           ADTAB    DC       BTE0&255          LO ADDRESS
7D0F       32                    DC       BTE1&255
7D10       3E                    DC       BTE2&255
7D11       4A                    DC       BTE3&255
7D12       56                    DC       BTE4&255
7D13       62                    DC       BTE5&255
7D14       6E                    DC       BTE6&255
7D15       7A                    DC       BTE7&255
                        *
                        ** -- LOCTAB CONTAINS ADDRESS OF USER-PROGRAM INSTRUCTION
                        *       WHERE WE PLACED THE BREAKPOINT IN THE FIRST PLACE.
                        *
7D16                    LOCTAB   DS       2*8               SPACE FOR 16 PCH/L PAIRS
                        *
                        ** -- BREAK-TABLE ENTRIES (BTE'S) --- *
                        *
7D26                    BTE0     DS       12                12-BYTES RESERVED
7D32                    BTE1     DS       12
7D3E                    BTE2     DS       12
7D4A                    BTE3     DS       12
7D56                    BTE4     DS       12
7D62                    BTE5     DS       12
7D6E                    BTE6     DS       12
7D7A                    BTE7     DS       12                ENOUGH FOR 8 BREAKPOINTS
                        *
                        * END OF DATA AREAS
                        *  THE REST IS ROM-ABLE.
                        *


                        ********************************************************************
                        *       NAME:    FINDPC
                        *       PURPOSE: CHECK IF PC IN FW1/FW2 MATCHES ANY IN LOCTAB
                        *       RETURNS: CARRY SET IF YES; XREG=ADTAB INDEX 0-7
                        *                CARRY CLR IF NOT; XREG=GARBAGE
                        *       VOLATILE:DESTROYS AC
                        ********************************************************************
7D86    A2 0F           FINDPC   LDXIM    15                BYTE-INDEX TO END OF TABLE
7D88    AD 04 7D        FPC00    LDA      FW2               GET FOR COMPARE
7D8F    DD 16 7D                 CMPX     LOCTAB            A PCH MATCH?
7D8E    D0 08                    BNE      FPC02             =>NO. TRY NEXT 2-BYTE ENTRY
7D90    AD 03 7D                 LDA      FW1               GET PCL NOW
7D93    DD 15 7D                 CMPX     LOCTAB-1          A PCL MATCH?
7D96    F0 06                    BEQ      FPC04             =>YES! WE HAVE A BREAKPOINT!
7D98    CA              FPC02    DEX                        BACK UP ONE
7D99    CA                       DEX                         AND ANOTHER
7D9A    10 EC                    BPL      FPC00             =>DO ENTIRE TABLE SCAN
7D9C    18                       CLC                        =>DONE; SCAN FAILED
7D9D    60                       RTS
                        *
7D9E    48              FPC04    PHA                        HOLD AC
7D9F    8A                       TXA                        HALVE VALUE IN XREG
7DA0    4A                       LSRA                        SINCE IT'S 2-BYTE INDEX
7DA1    AA                       TAX
7DA2    68                       PLA
7DA3    38                       SEC                        SET 'SUCCESS'
7DA4    60                       RTS


                        ********************************************************************
                        *       NAME:    BREAK
                        *       PURPOSE: HANDLE ENTRY AT BRK AND PROCESS BREAKPOINTS
                        *       NOTE:    THIS ROUTINE GETS ENTERED ON *EVERY* 'COUT'
                        *                CALL--IT KNOWS ABOUT BRK BECAUSE THE MONITOR'S
                        *                REGISTERS ARE SETUP TO PRINT USER REG CONTENTS.
                        *                AFTER PROCESSING IS DONE, IT RESTORES THE MONITOR'S
                        *                REGS AND RETURNS.
                        ********************************************************************
7DA5    E0 FB   BREAK    CPXIM    X'FB'             IS XREG SET FOR EXAMINE-REGS?
7DA7    D0 27            BNE      BRKXX             =>NO GET OUT NOW.
```

```
7DA9        C9 A0       BRK02   CMPIM   X'A0'       IS AC SETUP CORRECTLY TOO?
7DAB        D0 23               BNE     BRKXX       =>NOPE. FALSE ALARM!
7DAD        A5 3C               LDAZ    A1L         GET USER PCL
7DAF        38                  SEC                  AND BACK IT UP
7DB0        E9 02               SBCIM   2             BY 2 BYTES SINCE
7DB2        8D 03 7D            STA     FW1            BRK BUMPED IT!
7DB5        A5 3D               LDAZ    A1H         GET PCH
7DB7        E9 00               SBCIM   0           DO THE CARRY
7DB9        8D 04 7D            STA     FW2          AND SAVE THAT TOO
7DBC        20 86 7D            JSR     FINDPC      A BREAKER OF OURS HERE?
7DBF        90 0F               BCC     BRK04       =>NOPE. WE WON'T HANDLE IT!
7DC1        BD 0E 7D            LDAX    ADTAB       YES; GET BTE ADDRESS THEN
7DC4        8D 05 7D            STA     PCL          AND SET IT AS THE 'GO'
7DC7        A9 7D               LDAIM   LOWPAGE        PC FOR THE 'GO' COMMAND.
7DC9        8D 06 7D            STA     PCH         (OUR PAGE FOR BTE'S)
                        *
7DCC        A9 A0       BRK04   LDAIM   X'A0'       SET AC BACK FOR MONITOR
7DCE        A2 FF               LDXIM   X'FF'        AND XREG TOO
7DD0        4C F0 FD    BRKXX   JMP     X'FDF0'     =>NO. RIGHT BACK TO COUT ROUTINE!
```

```
                    ********************************************************************
                    ***       PROCESS THE 'GO' COMMAND (RESUME USER EXECUTION)  **
                    *          COMMAND FORMAT: ( * Yc G ) .
                    ********************************************************************
7DD3  AD 05 7D  CMDGO   LDA     PCL             GET RESUME PCL
7DD6  85 3C             STAZ    A1L              AND SETUP FOR MONITOR
7DD8  AD 06 7D          LDA     PCH              TO SIMULATE AN 'XXXX  G' COMMAND
7DDB  85 3D             STAZ    A1H                NORMALLY.
7DDD  4C B9 FF          JMP     X'FFB9'         =>SAIL INTO MONITOR'S 'GO'
```

```
                    ********************************************************************
                    **        WE GET CONTROL HERE ON THE CONTROL-Y USER EXIT FROM THE
                    *         MONITOR (ON KEYINS). ALL COMMANDS ARE SCANNED HERE;
                    *         CONTROL WILL PASS TO THE APPROPRIATE ROUTINE.
                    ********************************************************************
7DE0        A2 FF       KEYIN   LDXIM   X'FF'           CHAR INDEX
7DE2        E8          KEYIN00 INX                     SET NEXT CHARACTER
7DE3        BD 00 02            LDAX    X'0200'         GET CHAR FROM KEYIN BUFFER
7DE6        C9 99               CMPIM   X'99'           CONTROL-Y CHARACTER?
7DE8        D0 F8               BNE     KEYIN00         =>NO. KEEP SCANNING
7DEA        E8                  INX                     BUMP OVER CTL-Y
7DEB        BD 00 02            LDAX    X'0200'         GRAB COMMAND CHARACTER
7DEE        C9 C7               CMPIM   X'C7'           IS IT 'G' (GO) ?
                        *
                        * A BRANCH-TABLE WOULD BE
                        *  NEATER, BUT IT WOULD
                        *  TAKE UP MORE CODE FOR
                        *  THE FEW OPTIONS WE HAVE.
                        *
7DF0        F0 E1               BEQ     CMDGO           =>YES.
7DF2        C9 C1               CMPIM   X'C1'           IS IT 'A' (ADD) ?
7DF4        F0 18               BEQ     CMDADD          =>YES.
7DF6        C9 C4               CMPIM   X'C4'           IS IT 'D' (DISPLAY) ?
7DF8        F0 0B               BEQ     XXDISP          =>YES.
7DFA        C9 D2               CMPIM   X'D2'           IS IT 'R' (REMOVE) ?
7DFC        F0 0A               BEQ     XXREMOVE        =>YES.
7DFE        C9 C9               CMPIM   X'C9'           IS IT 'I' (INIT) ?
7E00        F0 09               BEQ     XXINIT          =>YES.
7E02        4C 65 FF    BADCMD  JMP     RESET           NOTHING; IGNORE IT!
                        *
7E05        4C A8 7E    XXDISP  JMP     CMDDISP         EXTENDED BRANCH
7E08        4C 08 7F    XXREMOVE JMP    CMDREMOV        EXTENDED BRANCH
7E0B        4C 4F 7F    XXINIT  JMP     CMDINIT         EXTENDED BRANCH
```

```
                        ********************************************************************
                        **       PROCESS THE 'ADD' COMMAND..ADD A BREAKPOINT AT
                        **        LOCATION SPECIEIED IN COMMAND
                        *          COMMAND FORMAT: ( * aaaa Yc A ) .
                        ********************************************************************
7E0E    A0 00       CMDADD  LDYIM   0               CHECK OPCODE FIRST
7E10    B1 3E               LDAIY   A2L             OP AT AAAA A BRK ALREADY?
7E12    F0 EE               BEQ     BADCMD          =>YES. ILLEGAL!
                        *
                        * --- SCAN LOCTAB FOR AN AVAILABLE BTE TO USE --- *
                        *
7E14    A2 0E               LDXIM   15              BYTE INDEX TO LOCTAB END
7E16    BD 16 7D    ADD00   LDAX    LOCTAB          GET A BYTE
7E19    D0 05               BNE     ADD02           =>IN USE
7E1B    BD 15 7D            LDAX    LOCTAB-1        GET HI HALE
7E1E    F0 06               BEQ     ADD04           => BOTH ZERO; USE IT!
7E20    CA          ADD02   DEX                     MOVE BACK TO
7E21    CA                  DEX                      NEXT LOCTAB ENTRY
7E22    10 E2               BPL     ADD00             AND KEEP TRYING!
7E24    30 DC               BMI     BADCMD          =>DONE? ALL EULL! REJECT IT.
                        *
7E26    A5 3E       ADD04   LDAZ    A2L             GET aaaa VALUE
7E28    9D 15 7D            STAX    LOCTAB-1        SAVE LO HALE
7E2B    8D 0B 7D            STA     SKEL+4          STUEE LO ADDR INTO BTE
7E2E    A5 3E               LDAZ    A2H             GET aaaa VALUE
7E30    9D 16 7D            STAX    LOCTAB          SAVE HI HALE
7E33    8D 0C 7D            STA     SKEL+5          STUEE HI ADDR INTO BTE
7E36    8A                  TXA                     GRAB INDEX EOR LOCTAB
7E37    4A                  LSRA                    MAKE ADTAB INDEX
7E38    AA                  TAX                      AND STUFF BACK INTO XREE
7E39    A9 7D               LDAIM   LOWPAGE         BTE'S HI ADDRESS VALUE
7E3B    85 41               STAZ    A3H             HOLD IN WORK AREA
7E3D    BD 0E 7D            LDAX    ADTAB           GET BTE LO ADDR EROM ADTAB
7E40    85 40               STAZ    A3L             SAVE IN WORK AREA
7E42    A0 07               LDYIM   7               7-BYTE MOVE FOR SKEL BTE
7E44    B9 07 7D    ADD06   LDAY    SKEL            GET SKEL BYTE
7E47    91 40               STAIY   A3L             MOVE TO BTE
7E49    88                  DEY                     SET NEXT
7E4A    10 E8               BPL     ADD06           => MOVE ENTIRE SKELETON
7E4C    C8                  INY
7E4D    B1 3E               LDAIY   A2L             GET ORIGINAL OPCODE
7E4E    91 40               STAIY   A3L              INTO BTE
7E51    20 8E E8            JSR     INSDS2          INSDS2 (TO DISASSEMBLE)
7E54    A9 00               LDAIM   0               SET BRK OPCODE
7E56    91 3E               STAIY   A2L              OVER ORIGINAL OPCODE
7E58    A5 2F               LDAZ    LENGTH          GET INSTRUCTION LENGTH
7E5A    38                  SEC
                        *
                        * --- SET UP JMP TO NEXT INST. IN THE BTE --- *
                        *
7E5B    A0 04               LDYIM   4
7E5D    71 40               ADCIY   A3L             ADD TO PC FOR DESTINATION
7E5F    91 40               STAIY   A3L             STUFE INTO BTE
7E61    C8                  INY
7E62    B1 40               LDAIY   A3L             RUN UP THE CARRY
7E64    69 00               ADCIM   0                RIGHT HERE
```

```
7E66    91 40               STAIY   A3L             STUFF ADDRESS INTO JMP
7E68    A5 2E               LDAZ    FORMAT          GET INSTRUCTION FORMAT
7E6A    C9 9D               CMPIM   X'9D'           IS FORMAT=BRANCH?
7E6C    F0 16               BEQ     ADDBRCH         =>YES. MORE TO DO
7E6E    A5 2F               LDAZ    LENGTH          LENGTH=1?
7E70    F0 0F               BEQ     CMDRET          =>YES. DONE
7E72    6A                  RORA                    LENGTH=2?
7E73    B0 06               BCS     ADDLEN2         =>YES
7E75    A0 02               LDYIM   2               LENGTH=3;MOVE 3RD BYTE TO BTE
7E77    B1 3E               LDAIY   A2L             GET INST 3RD BYTE
7E79    91 40               STAIY   A3L              AND MOVE TO BTE
7E7B    A0 01       ADDLEN2 LDYIM   1               LENGTH=2;MOVE 2ND BYTE TO BTE
7E7D    B1 3E               LDAIY   A2L             GET INST 2ND BYTE
7E7F    91 40               STAIY   A3L              AND MOVE TO BTE
7E81    4C 69 FF    CMDRET  JMP     MON             DONE; BACK TO MONITOR!
```

```
                        *
                        * --- FOR BRANCHES, WE'VE GOTTA ADD A JMP FOR THE 'TRUE'
                        *       CONDITION (SINCE WE MOVED THE BRANCH 'WAY OUTA THE PROGRAM!)
                        *
7E84      A0 01         ADDBRCH LDYIM   1               SET FOR 2ND BYTE
7E86      B1 3E                 LDAIY   A2L             GET DESTINATION OFFSET
7E88      18                    CLC                      AND ADD 2 BYTES TO
7E89      69 02                 ADCIM   2                 CONSTRUCT ABS ADDRESS
7E8B      65 3E                 ADCZ    A2L             ADD TO SUBJECT-INST ADDRESS
7E8D      85 3E                 STAZ    A2L
7E8F      A5 3F                 LDAZ    A2H             CARRY IT
7E91      69 00                 ADCIM   0
7E93      85 3F                 STAZ    A2H
7E95      EA                    NOP                     (PLACE-HOLDER WASTE HERE)
7E96      A9 04                 LDAIM   4               TRUE-BRANCH TO +4
7E98      91 40                 STAIY   A3L             PUT INTO NEW OFFSET
7E9A      A0 07                 LDYIM   7
7E9C      A5 3E                 LDAZ    A2L             GET JMP ADDRESS
7E9E      91 40                 STAIY   A3L             MOVE IT TO
7EA0      C8                    INY                      THE
7EA1      A5 3F                 LDAZ    A2H               BTE FOR
7EA3      91 40                 STAIY   A3L                THE 'TRUE' JMP
7EA5      B8                    CLV                     SNEAKY BRANCH
7EA6      50 D9                 BVC     CMDRET           TO EXIT
7EBA       8A         DISP04   TXA                           GET INDEX
7EBB       48                  PHA                            SAVE IT
7EBC       BC 16 7D            LDYX    LOCTAB                GET SUBJECT-INST PCH
7EBF       BD 15 7D            LDAX    LOCTAB-1               AND ITS PCL
7EC2       84 3B               STYZ    X'3B'                 SET UP PCH/PCL FOR
7EC4       85 3A               STAZ    X'3A'                  DISASSEMBLER...
7EC6       AA                  TAX
7EC7       20 40 F9            JSR     PRNTYX                PRINT Y,X BYTES IN HEX
7ECA       A9 A0               LDAIM   X'A0'                 PRINT ONE
7ECC       20 ED FD            JSR     COUT                   SPACE HERE
7ECF       68                  PLA                           RESTORE INDEX
7ED0       48                  PHA
7ED1       4A                  LSRA                          CONVERT TO ADTAB INEX
7ED2       AA                  TAX
7ED3       A9 BC               LDAIM   X'BC'                 '<' CHARACTER
7ED5       20 ED FD            JSR     COUT                  PRINT IT
7ED8       A9 7D               LDAIM   LOWPAGE               BTE HI ADDRESS
7EDA       85 3F               STAZ    A2H                   SET INDIRECT POINTER
7EDC       20 DA FD            JSR     PRBYTE                PRINT HEX BYTE
7EDF       BD 0E 7D            LDAX    ADTAB                 GET BTE LO ADDR
7EE2       85 3E               STAZ    A2L                   SET INDIRECT POINTER
7EE4       20 DA FD            JSR     PRBYTE                PRINT BTE FULL ADDRESS
7EE7       A9 BE               LDAIM   X'BE'                 '>' CHARACTER
7EE9       20 ED FD            JSR     COUT                  PRINT IT
                        *
                        * --- DISASSEMBLE THE ORIGINAL INSTRUCTION. PICK UP
                        *       ORIGINAL OPCODE FROM BTE, ORIGINAL ADDRESS
                        *       FIELD FROM USER PROGRAM LOCATION.
                        *
7EEC      A9 A0               LDAIM   X'A0'                 PRINT ONE
7EEE      20 ED FD            JSR     COUT                   SPACE HERE
7EF1      A0 00               LDYIM   0                     INDEX
7EF3      B1 3E               LDAIY   A2L                   GET OPCODE FROM BTE
7EF5      20 DA FD            JSR     PREYTE                PRINT OPCODE
7EF8      B1 3E               LDAIY   A2L                   GET OPCODE FROM BTE
7EFA      20 8E F8            JSR     INSDS2                 AND GET FORMAT/LENGTH
7EFD      20 04 7F            JSR     JSRKLUGE              SNEAK INTO INSDSP @ F8D9
7F00      68                  PLA
7F01      AA                  TAX                           RESTORE LOCTAB INDEX
7F02      10 B0               BPL     DISPNXT               => DISPLAY THE REST!
                        *
```

```
                        * KLUGE ENTRY INTO SUBROUTINE
                        *  WHICH FORCES JSR PRIOR TO
                        *   A PHA INSTRUCTION. WE HAVE
                        *    TO JSR TO THIS JMP!
                        *
7F04        48          JSRKLUGE PHA                        PUSH MNEMONIC INDEX
7F05        4C D9 F8                 JMP      X'F8D9'       CONTINUE WITH INSTDSP
                        ******** END OF KLUGE! *****


                        ********************************************************************
                        *       REMOVE A BREAKPOINT AT LOCATION aaaa
                        *        COMMAND FORMAT: ( aaaa Yc R )
                        ********************************************************************
7F08        A5 3E       CMDREMOV LDAZ     A2L               GET ADDRESS LO
7F0A        8D 03 7D             STA      FW1               HOLD IT FOR FINDPC
7F0D        A5 3F                LDAZ     A2H               GET ADDRESS HI
7F0F        8D 04 7D             STA      FW2
7F12        20 86 7D             JSR      FINDPC            A BREAKPOINT HERE?
7F15        B0 03                BCS      REMOV02           =>YES
7F17        4C 65 FF             JMP      RESET             =>NO; BELL FOR YOU!
                        *
7F1A        BD 0E 7D    REMOV02  LDAX     ADTAB             GET THE LOCTAB ENTRY
7F1D        85 40                STAZ     A3L               HOLD IT
7F1F        8A                   TXA                        NOW CREATE LOCTAB INDEX
7F20        0A                   ASLA
7F21        AA                   TAX
7F22        A9 00                LDAIM    0                 CLEAR OUT THE
7F24        A8                   TAY                         APPROPRIATE
7F25        9D 16 7D             STAX     LOCTAB              LOCTAB ENTRY
7F28        9D 17 7D             STAX     LOCTAB+1             FOR THIS BKPT
7F2B        A9 7D                LDAIM    LOWPAGE           HI ADDR FOR BYTE
7F2D        85 41                STAZ     A3H               HOLD FOR ADDRESSING
7F2F        B1 40                LDAIY    A3L               GET OPCODE OUT OF BYTE
7F31        91 3E                STAIY    A2L                AND PUT BACK INTO ORIGINAL
7F33        4C 69 FF             JMP      MON               =>ALL DONE.


                        ********************************************************************
                        *        INITIALIZATION CODE. ENTERED AT START ADDR TO INITIALIZE.
                        *      IT CLEARS LOCTAB, SETS UP THE Yc AND 'COUT' EXITS.
                        *
                        *        AFTER EVERY 'RESET', MUST RESETUP WITH * Yc I .
                        ********************************************************************
7F36    A9 4C       INITX    LDAIM    X'4C'             JMP OPCODE
7F38    8D F8 03             STA      X'3F8'            STUFF IN Yc EXIT LOC
7F3B    A9 7D                LDAIM    KEYIN/256         KEYIN: HI ADDRESS
7F3D    8D FA 03             STA      X'3FA'            STUFF INTO JMP
7F40    A9 E0                LDAIM    KEYIN&X'FF'       KEYIN: LO ADDRESS
7F42    8D F9 03             STA      X'3F9'            STUFF INTO JMP ADDRESS
7F45    A9 00                LDAIM    0
7F47    A2 0F                LDXIM    15                INDEX TO LOCTAB END
7F49    9D 16 7D    INIT00   STAX     LOCTAB            CLEAR IT OUT
7F4C    CA                   DEX                         SO THERE ARE
7F4D    10 FA                BPL      INIT00             NO BREAKPOINTS
                        *
                        * ---- ENTER HERE AFTER HITTING 'RESET' KEY, PLEASE --- *
                        *
7F4F    A9 A5       CMDINIT  LDAIM    BREAK&255         BREAK: LO ADDRESS
7F51    85 36                STAZ     CSWL              STUFF INTO 'COUT' EXIT HOOK
7F53    A9 7D                LDAIM    BREAK/256         BREAK: HI ADDRESS
7F55    85 37                STAZ     CSWH              STUFF INTO 'COUT' EXIT HOOK
7F57    4C 69 FF             JMP      MON               INIT DONE; BACK TO MON.
                             END
```

TWO APPLE II ASSEMBLERS: A COMPARATIVE SOFTWARE REVIEW

Allen Watson
430 Lakeview Way
Redwood City, CA 94062

There are two assembler programs for the Apple II available from independent software vendors: the Microproducts Apple II Co-resident Assembler for $19.95 from Microproducts, 1024 17th Street, Hermosa Beach, CA 90254, and the S-C Assembler II for $25 from S-C Software, P.O. Box 5537, Richardson, TX 75080. The features and relative merits of these assemblers are the subject of this review.

Introduction: Software Tools

Some microcomputer owners hardly ever program, being satisfied to run programs written by other people. Others program only in BASIC or one of the compiler languages. Then there are those who write programs in machine language because the demands they make of their computers can be met in no other way. The assembler is a software tool which relieves them of much of the drudge-work involved in machine-language programming.

Software tools such as assemblers are much more important than their modest sizes might imply, since they are used over and over in the development of other programs. A poor tool is tiring to use and causes errors and frustration; a good tool requires minimum effort and soon seems like a natural extension of the user.

Built-In Assembler Features

The mini-assembler built into the Apple II sets it apart from conventional microcomputers. It will probably lead many Apple II owners to venture into machine-language programming for the first time.

The mini-assembler's primary function is instruction-code translation. Instead of remembering all the 6502 numeric opcodes, the programmer finds himself thinking in the 6502 mnemonics. The word **mnemonic** just means **easy to remember**; while letter combinations such as CMP and LDA may seem cryptic at first, it soon becomes second-nature to read CMP as **compare** and LDA as **load accumulator**.

The branch instructions in the 6502 use relative addresses. The address that is being branched to has to be converted into a one-byte offset value. Doing this by hand is so tedious and prone to error that there is even a small slide rule on the market to do the hexadecimal arithmetic. The Apple's mini-assembler and its companion disassembler take care of this automatically, so that the programmer can use the actual address values when he writes branch instructions.

The different addressing modes of the 6502 are handled very simply. Indexing is indicated by a comma and X or Y after the base address. Parentheses are used to delimit the address of the address in indirect-addressing mode, and indirect-indexed and indexed-indirect addressing are easily distinguished by this means.

The Apple's built-in assembler is very convenient, but the machine could do more for him. Obviously, given the right program, it can. Enter the full-fledged assemblers, stage right.

More Assembler Features

Both of the assemblers described here have all the features of the Apple mini-assembler and several more besides. The two most important additional features are program editing and symbolic addressing. An editor is often a separate program, but since much of the value of an assembler would be lost without the ability to edit, both of these assemblers include editors and should properly be called editor-assemblers.

Once you face the necessity of re-entering most of a long program by hand in order to make room for additional instructions near the beginning of the program the need for an editor will be apparent. Some machines have editors that work directly on the machine code, but the editor portions of both of these assemblers manipulate the assembler input data or source file. They enable the programmer to add or delete instructions anywhere in the program without worrying about the consequences. (Well, almost; if the added instructions between a branch instruction and its destination increase the displacement to more than 128 bytes, the branch is no longer valid and must be replaced by a different branch and a jump.)

Symbolic addressing is one of the most important functions of an assembler. The older higher-level language BASIC and FORTRAN have symbolic addressing only for variables. The lack of symbolic addressing of instructions makes programs difficult to read.

Address references in assembler language are made by means of symbols which are assigned their numeric values when the program is assembled. The programmer needn't be concerned about the actual addresses except to make sure there is room for all of them. But symbolic addressing does more than just eliminate a lot of messy bookkeeping: since the symbols are entirely arbitrary, the programmer can choose them such that they serve as mnemonic labels for all of the important addresses in the program. For example, where a BASIC programmer would have to write something like GOTO 1275, an assembler-language programmer may write JMP DONE, where DONE is both a symbol which represents the required address and a label which is meaningful to the programmer.

The Microproducts Co-resident Assembler and the S-C Assembler II both qualify as full-fledge assemblers. They have several features in addition to those described above, including:

(1) loading and saving the assembler input file on tape;
(2) programmer specification of the starting address in memory of the assembled program;
(3) inclusion of ASCII character strings and hexadecimal numbers as part of the program; and
(4) the inclusion of comments, explanatory notes which are part of the input file but are ignored by the assembler.

What About Documentation?

A user's manual is provided with each of these assemblers. The Microproducts manual consists of seven pages and is barely adequate. It is poorly organized and there are a couple of errors in it. The manual for the S-C assembler is more substantial, with 17 pages of instructions giving complete information for the programmer. There are also 10 pages of appendices including a list of references and a listing of a printer-driver program. It is clear and candid, even pointing out a couple of weak places in the program.

Now For The Bad News

There are limits to how easy things can be made for the machine-language programmer. For one thing, both assemblers limit the length of symbols to not more than four characters, and special characters are not permitted: only letters and numbers. Another joy-killer is the strict formatting of the input statements. Labels must be in their specified columns, opcodes in theirs, and so on. If there is no label on a particular line, you must skip across to the correct column before typing in the operation mnemonic.

The S-C assembler ameliorates this problem by providing a tabulation feature: to skip a field, you just type in a TAB. Since the Apple II's keyboard doesn't have a TAB key, you have to use Control-I for this. The Microproducts assembler makes you count spaces, which is downright criminal. Computers can count without ever making a mistake, but programmers can't; therefore programmers should never be called upon to count when there is a computer available to do it for them.

Editing With Line Numbers

Both of these assemblers include editors that work like the BASIC editor by using line numbers. The programmer must type a line number at the beginning of every line, and the sequence of the numbers becomes the sequence of the lines. And woe be unto him who accidentally uses the same numbers twice: the lines entered earlier will be written over by the later ones having the same numbers. If you have never been so careless as to make this error, reading about it here will probably suggest it to your subconscious, so beware!

Now suppose that you have just typed in a program that is 250 lines long, dutifully numbering the lines in steps of 10, and you want to examine an earlier part of the program. What do you do? If you have a printer, you can list the whole thing and examine any part you want to. Both assemblers include commands for starting and stopping a printer. But short of listing the whole program, suppose you just want to display part of it on the TV screen.

Either assembler will enable you to start through the whole input file on the TV display and interrupt it when you reach the desired part, that is, if you have fast reactions. The S-C program is kinder: it has a SLOW mode for displaying. It also lets you specify range of line numbers to display, just as you do in BASIC.

The S-C assembler has another feature which should prove very useful: you can APPEND a source file saved on tape earlier onto the input file you are currently editing in memory and assemble the whole thing as a single program. This makes it possible to build yourself a library of standard routines which you can use in several different programs with a minimum of effort.

Shortcomings of the Microproducts Assembler

There aren't a great many nice things I can say about the Microproducts assembler. It simply doesn't do all the things it should to help the programmer. For example, error messages are output as number codes which you have to look up in the manual. If it were programmed to do so, the computer could look them up a lot faster and put them out in English. With the S-C assembler, it does.

In the Microproducts version, numeric expressions must include leading zeros. If you define a symbol as RATE .DL 5, RATE will be assembled as hexadecimal 5000, not 0005. But what's even more exasperating, once you get it defined as 0005, references to RATE will not assemble as zero-page addressing unless you prefix the symbol with an asterisk each time it is referenced. This is plain inexcuseable: the program should test for this and select the appropriate address mode automatically.

Are There Bugs in the Programs?

Nobody's perfect, not even the people who write assemblers. No matter how hard they try, debugging can't demonstrate the absence of bugs, only their presence. While I haven't tried out every feature of these assemblers yet, I have assembled the same program on both of them as a comparison. So far I have found only one bug in the S-C assembler. If you slip while typing an implied-operand instruction without a label and put the mnemonic in the label columns thus leaving the operation and operand fields blank the assembler will not detect the error but instead will repeat the previous instruction.

The Microproducts assembler has bugs, too. It permits a comment on an instruction line, but if the comment is long enough that the line exceeds 40 columns so that the display continues on a second line, the address and object code which normally appear at the left of the screen get written on the second line and obliterate the comment. Another bug appears whenever you interrupt a listing, which you can do by hitting any key. The Microproducts assembler fails to clear the keyboard strobe, causing the key you used to interrupt it to become the first character of the next command.

There is a curious error in the Microproducts manual where it states that the assembler is less than 3K bytes long, even though it loads from 2000 to 2CFF in memory, a total of 3,328 bytes. Just coincidentally, the S-C assembler loads from 1000 to 1BFF, making it exactly 3K bytes long.

Wouldn't It Be Nice If...?

While both of these assemblers are more powerful than the mini-assembler, some people are never satisfied. A couple of improvements occurred to me as soon as I started using these assemblers.

In a BASIC program, the line numbers are an innate part of the program, used as destinations for GOTOs and so on. Assembler language doesn't really use line numbers; these assemblers use them only because they make the editor simpler. It would be nice if the programmer didn't have to keep track of a lot of numbers; the computer is much better at it. If the editor has to have line numbers, an automatic line-number generator would be a nice option.

I'd like to see some kind of LOCATE function, too. Since the line numbers don't bear much relation to the program, especially after you've used the RENUMBER a time or two, the selective list feature of the S-C assembler isn't 100% effective for displaying a portion of the program. What if you don't remember the line number of the instruction you labelled SCAN? Wouldn't it be nice if you could type something like LOCATE "SCAN" and have the editor search for the line that has SCAN as its label? Some editors even have two different forms of this command: one which looks only at the beginning of each line, and another which searches all the way through each line to find the places where a label is used in an operand or in a comment.

It is interesting to note the similarities between these two assemblers. The programs are nearly the same size, about 3K bytes, and priced at $20-$25. They use similar input formats and both of them do their editing by means of BASIC-type line numbers.

Where they diverge the advantage is almost always with the S-C Assembler II. It has more features and a bigger manual, its error messages are output in English, and its format is a more logical extension of the Apple II mini-assembler. If you are the least bit interested in machine-language programming on the Apple II, I strongly recommend the purchase of a copy of the S-C Assembler II.

APPLE CALLS AND HEX-DECIMAL CONVERSION

Marc Schwartz
220 Everit Street
New Haven, CT 06511

Rich Auricchio's "Programmer's Guide to the Apple II" (MICRO #4, April/May 1978) is a very useful step in getting out printed materials to help users fully exploit the Apple's potential. That his table of monitor routines can be used in BASIC programming is worth noting.

Many monitor routines can be accessed in BASIC by CALL commands addressed to the location of the first step of the routine. If the routine is located in hex locations 0000 to 4000, it is necessary only to convert the hex location to decimal and write CALL before the decimal number. Thus a routine located at hex 1E would be accessed by the command: CALL 30, since hex 001E = decimal 30.

If you do not have a hex-decimal conversion table handy, you can convert larger numbers to decimal with the help of the Apple by the following steps:

1. Start in BASIC (necessary for step 2)
2. Multiply the first (of four) hex digits by 4096, the second by 256, the third by 16 and the fourth by one. Add the four numbers to get the decimal equivalent. For example, to get the decimal conversion of 03E7, with the Apple in BASIC, press Control/C and type

```
>PRINT 0*4096 + 3*256 + 14*16 + 7
```

then press RETURN. You'll get your decimal answer: 839. To begin a monitor routine you wrote starting at 03E7, merely put CALL 839 in your program.

If the hex location of the routine is between C000 and FFFF, then another method of figuring out the corresponding decimal location must be used.

1. Start in BASIC
2. Press the RESET button.
3. Take the hex location of the routine and subtract if from FFFF. The Apple will help you do this; subtract each pair of hex digits from FF and press RETURN. The Apple will print the answer to each subtraction for you. For example the hex location of the routine to home cursor and clear screen is $FC58.

```
* FF - FC RETURN
  = 03
* FF - 58 RETURN
  = A7
```

So, $FFFF - $FC58 = $03A7.

Now convert to decimal as above, using BASIC (control/C) to assist you.

```
>PRINT 0*4096 + 3*256 + 10*16 + 7
```

and after pressing RETURN you will have your answer, 935.

4. Add one to the total, here giving 936.
5. Make the new total negative, or -936.
6. That's it. Now just put a CALL in front of the number: CALL -936.

Of course, these steps of converting hex locations to decimal are the same ones to take if you want to access the PEEK or POKE functions of the Apple. In all, they allow the BASIC programmer to take much fuller advantage of the capabilities of the computer.

And while on the subject of hex-decimal conversion, the Apple can help in decimal to hex conversion as well. For example to find the hex of a number, say 8765:

1. Start in BASIC
2. Divide the number by 4096, then find the remainder:

```
>PRINT 8765/4096,8765MOD4096 (return)
 2        573
```

3. Repeat the process with 256 and16:

```
>PRINT 573/256,573MOD256 (return)
 2        61
>PRINT 61/16, 61 MOD 16 (return)
 3         13
```

...giving 2 2 3 13 or 223C.

APPLE II HIGH RESOLUTION GRAPHICS
MEMORY ORGANIZATION

Andrew H. Eliason
28 Charles Lane
Falmouth, MA 02540

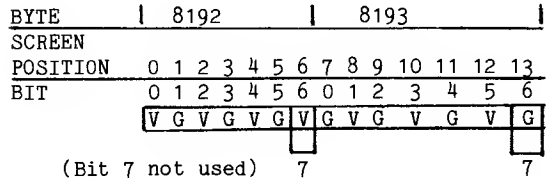
One of the most interesting, though neglected, features fo the Apple II computer is its ability to plot on the television screen in a high resolution mode. In this mode, the computer can plot lines, points and shapes on the TV display area in greater detail than is possible in the color graphics mode (GR) which has a resolution of 40 x 48 maximum.

In the high resolution (HIRES) mode, the computer can plot to any point within a display area 280 points wide and 192 points high. While this resolution may not seem impressive to those who have used plotters and displays capable of plotting hundreds of units per inch, it is nonetheless capable of producing a very complex graphic presentation. This may be easily visualized by considering that a full screen display of 24 lines of 40 characters is "plotted" at the same resolution. An excellent example of the HIRES capability is included in current Apple II advertisements.

Why, then, has reletively little software appeared that uses the HIRES features? One of the reasons may be that little information has been available regarding the structure and placement of words in memory which are interpreted by HIRES hardware. Information essential to the user who wishes to augment the Apple HIRES routines with his own, or to explore the plotting possibilities directly from BASIC. In a fit of curiosity and Apple-insomnia, I have PEEKed and POKEd around in the HIRES memory area. The following is a summary of my findings. Happy plotting!

Each page of HIRES Graphics Memory contains 8192 bytes. Seven bits of each byte are used to indicate a single screen position per bit in a matrix of 280H x 192V. The eighth bit of each byte is not used in HIRES and the last eight bytes of every 128 are not used.

The bits in each byte and the bytes in each group are plotted in ascending order in the following manner. First consider the first two bytes of page 1. (Page 2 is available only in machines with at least 24K).



V = VIOLET
G = GREEN

Figure 1 represents the screen position and respective bit & word positions for the first 14 plot positions of the first horizontal line. If the bit is set to 1 then the color within the block will be plotted at the position indicated. If the bit is zero, then black will be plotted at the indicated position. It can be seen that even bits in even bytes plot violet, even bits in odd bytes plot green and vice versa. Thus all even horizontal positions plot violet and all odd horizontal politions plot green. To plot a single white point, one must plot the next higher or lower horizontal position along with the point, so that the additive color produced is white. This is also true when plotting single vertical lines.

The memory organization for HIRES is, for design and programming considerations, as follows:

Starting at the first word, the first 40 bytes (0-39) represent the top line of the screen (40 bytes x 7 bits = 280). The next 40 bytes, however, represent the 65th line (i.e., vertical position 64). The next 40 bytes represent the line at position 128 and the next 8 bytes are ignored. The next group of 128 bytes represent three lines at positions 8, 72 and 136, the next group at positions 16, 80 and 142, and so on until 1024 bytes have been used. The next 1024 bytes represent the line starting at vertical position 1 (second line down) in the same manner. Eight groups of 1024 represent the entire screen. The following simple porgram provides a good graphic presentation as an aid to understanding the above description. Note that there is no need to load the HIRES machine language routines with this program. Set HIMEM:8191 before you type in the program.

```
100 REM  SET HIMEM:8191
110 REM  HIRES GRAPHICS LEARNING AID
120 POKE -16304,0: REM  SET GRAPHICS MODE
130 POKE -16297,0: REM  SET HIRES MODE
140 REM   CLEAR PAGE - TAKES 20 SECONDS
150 FOR I=8192 TO 16383: POKE I,0: NEXT I
160 INPUT "ENTER BYTE (1 to 127)", BYTE
170 POKE -16302,0: REM  CLEAR MIXED GRAPHICS
180 FOR J=8192 TO 16383: REM  ADDRESS'
190 POKE J,BYTE: REM  DEPOSIT BYTE IN ADDRESS
200 NEXT J
210 POKE -16301,0: REM  SET MIXED GRAPHICS
220 GOTO 160
999 END
```

An understanding of the above, along with the following equations will allow you to supplement the HIRES graphics routines for memory efficient programming of such things as: target games, 3D plot with hidden line supression and 3D rotation, simulation of the low resolution C=SCRN (X,Y) function, etc. Also, you may want to do some clever programming to put Flags, etc., in the unused 8128 bits and 512 bytes of memory!

HI RES Graphics Equations and Algorithms

Where:

FB = ADDRESS OF FIRST BYTE OF PAGE.
PAGE1 = 8192 PAGE 2 = 16384
LH = HORIZONTAL PLOT COORDINATE. 0 TO 279
LV = VERTICAL PLOT COORDINATE. 0 TO 191
BV = ADDRESS OF FIRST BYTE IN THE LINE OF 40
BY = ADDRESS OF THE BYTE WITHIN THE LINE AT BV
BI = VALUE OF THE BIT WITHIN THE BYTE WHICH CORRESPONDS TO THE EXACT POINT TO BE PLOTTED.

Given: FB,LH,LV
BV = LV MOD 8 * 1024 + (LV/8) MOD 8 * 128 + (LN/64) * 40 + FB
BY = LH/7 + BV
BI = 2^(LH MOD 7)

To Plot a Point (Without HIRES Plot Routine):

```
LH = X MOD 280 : LV = Y MOD 192 (OR)
                 LV = 192-Y MOD 192
FB = 8192
BV = LV MOD 8 * 1024 + (LV/8) MOD 8 * 128 +
     (LV/64) * 40 + FB
BY = LH/7 + BV
BI = 2^(LH MOD 7)
WO = PEEK (BY)
IF (WO/BI) MOD 2 THEN (LINE NUMBER + 2)
POKE BY, BI + WO
RETURN
```

To Remove a Point, Substitute:

```
IF (WO/BI) MOD 2 = 0 THEN (LINE NUMBER + 2)
POKE BY, WO-BI
```

To Test a Point for Validity, the Statement:

"IF (WO/BI) MOD 2" IS TRUE FOR A PLOTTED POINT AND FALSE (=0) FOR A NON PLOTTED POINT.

MOS 16K RAM FOR THE APPLE II

Allen Watson III
430 Lakeview Way
Redwood City, CA 94062

MOS 16K dynamic RAM is getting cheaper. At the time of this writing, one mail-order house is offering 16K bytes of RAM (eight devices) for $120. Apple II owners can now enhance their systems for less than the Apple dealers' price. However, there is a potential drawback to the purchase of your own 16K RAM chips: speed. You may wonder why, since the Apple's 6502 CPU is running at only about 1 MHz, but things aren't quite that simple.

To begin with, the Apple II continually refreshes its video display and dynamic RAM. It does this by sharing every cycle between the CPU and the refresh circuitry, a half-cycle for each. This means that the RAM is being accessed at a 2 MHz rate.

That doesn't sound too fast, with the slowest 16K parts rated at 300ns access time; but you have to remember that the RAM chips are 16-pin parts by virtue of a multiplexed address bus. There are two address-strobe signals during each memory access cycle, and the access-time specification will be met only if the delay between these strobe signals is within specified limits. In the Apple II this delay is 140ns, which is too long. Furthermore, the Apple II timing doesn't allow long enough RAS precharge or row-address hold time for the slow parts. Judging by the spec sheets, 200ns parts are preferable to 250ns parts, and 300ns parts shouldn't be used at all. In my Apple, 300ns parts caused a zero to turn into a one once in a while.

Many mail-order houses do not mention device speeds in their ads. The best thing to do is to deal only with those suppliers who specify speeds, but for those who didn't, the table below shows the codes used by some 16K dynamic RAM manufacturers to indicate the speeds of their devices. Good luck, and caveat emptor!

SPEED CODES USED BY 16K DYNAMIC RAM MANUFACTURERS

		Access Time (ns)			
Manufacturer	Part No.	150	200	250	300
A M D	9016	-F	-E	-D	-C
Fairchild	F16K	-2	-3	-4	-5
Intel	2117	-2	-3	-4	
MOSTEK	4116	-2	-3	-4	
Motorola	MCM4116C	-15	-20	-25	-30
National	MM5290	-2	-3	-4	
N E C	μD416	-3	-2	-1	
T I	4116	-15	-20	-25	
Zilog	Z6166	-2	-3	-4	

LIFE FOR YOUR APPLE

Richard F. Suitor
166 Tremont St.
Newton, MA 02158

A listing of LIFE for the APPLE II is described briefly here (see MICRO #5 for a pet version and discussions). Because my experience with generation time in BASIC paralleled Dr. Covitz', the generation calculations are in assembly language. The display is initiated in BASIC and the routines are called from BASIC, which will slow down the generation time if desired.

The entire (40x48) low resolution graphics display is used. An unoccupied cell is 0 (black). An occupied one is 11 (pink). During the first half of a generation, cells that will die are set to color 8 (brown). Those to be born are set to color 3 (violet). During this stage, bit 3 set indicates a cell is alive this generation; bits 0 and 1 set indicate a cell will be alive the next). During the second half (mop-up) part those with bits 0 set are set alive (color 11), the rest are set to zero.

The BASIC program allows one to set individual cells alive, and to set randomly 1 in N alive in a rectangular region. The boundries (X = 0 and 39; Y = 0 and 47) do not change, but may be initialized. At the start of the program, NO PADDLE INTERVAL? is requested. If during the program the paddle reads close to 255 (as it will if none is connected) the number input here will be used instead. Zero is fastest, several generations per second. Entering 200 gives a few seconds per generation.

When X and Y coordinates are requested, put in the coordinates for any cells to be set alive. A negative X terminates this phase. Setting X=N and a negative Y will initialize a rectangular region to 1 in N randomly occupied and terminate the initialization. The boundaries of the rectangular region must be input and may be anywhere in the full display. A glider gun can be fit vertically in the display. However, don't initialize for Y 40 (other than random) for the scrolling during initialization input will wipe it out.

Before RUNning the BASIC program, set LOMEM: 2500 to avoid overwriting the subroutines.

```
>LIST
    1 TEXT
    2 GEN=2088
    3 MOP=2265
    5 DIM A$(7)
    7 K1=1
    8 K2=1
   10 CALL -936: VTAB 5: TAB 9: PRINT
      "CONWAY'S GAME OF LIFE"
   30 VTAB 15: PRINT "INITIATE PATTERN
       BELOW.  X<0 WILL START"
   35 PRINT "THE LIFE PROCESS.  A Y<0
      WILL GIVE A"
   40 PRINT "RANDOM PATTERN WITH ONE I
      N X ALIVE"
   50 VTAB 22: INPUT "RETURN TO CONTIN
      UE",A$
   99 GOTO 1000
  100 REM
  102 POKE -16302,0
  103 GOTO 130
  104 FOR I=1 TO K3
  105 CALL GEN
  107 FOR K=1 TO K1: NEXT K
  110 CALL MOP
  112 FOR K=1 TO K2: NEXT K
  120 NEXT I
  130 REM
  131 KX= PDL (0)-10
  132 IF KX>240 THEN KX=KX1
  135 IF KX<0 THEN KX=0
  140 K1=KX*6
  150 K2=KX*2
  155 K3=500/(K1+50)+1
  160 GOTO 104
 1000 GR
 1010 CALL -936
 1020 INPUT "NO PADDLE TIME INTERVAL "
      ,KX1
 1100 COLOR=11: INPUT "INPUT X,Y "
      ,X,Y
 1105 IF Y<0 THEN 1800
 1110 IF X<0 OR Y<0 THEN 2500
 1120 IF X>39 OR Y>39 THEN 1100
 1130 PLOT X,Y: GOTO 1100
 1800 INPUT "X DIRECTION LIMITS "
      ,I1,I2
 1810 IF I1<0 OR I2>39 OR I1>I2 THEN
      1800
 1820 INPUT "Y DIRECTION LIMITS "
      ,J1,J2
 1830 IF J1<0 OR J2>47 OR J1>J2 THEN
      1820
 2000 CALL -936: GR
 2001 POKE -16302,0
 2002 CALL -1998
 2005 FOR I=I1 TO I2
 2010 FOR J=J1 TO J2: COLOR=11: IF
      RND (X) THEN COLOR=0
 2020 PLOT I,J
 2030 NEXT J
 2040 NEXT I
 2100 GOTO 100
 2500 POKE -16302,0
 2510 COLOR=0
 2520 FOR K=40 TO 47
 2530 HLIN 0,39 AT K
 2540 NEXT K
 2590 GOTO 100
 9000 END
```

```
!A

                0010  :LIFE ROUTINES
                0020  :ENTER AT GEN0 AND MOP0 ALTERNATELY
                0030  :2088 AND 2265 DEC. RESP.
                0040  OLLN .DL 0002  OLD HORIZ LINE
                0050  NWLN .DL 0004  NEW LINE
                0060  SUM1 .DL 0006  # OF OCC. CELLS IN 3X3
                0070  SUM2 .DL 0007  1,2 FOR OLD,NEW
                0080  BUF1 .DL 0940  40 VERT. OCC. #S
                0090  BF1P .DL 0942
                0100  BF1M .DL 093F
                0110  BUF2 .DL 0970
                0120  BF2P .DL 0972
                0130  BF2M .DL 096F
0800  A505      0140  NXLN LDA *NWLN+01
0802  8503      0150       STA *OLLN+01
0804  A504      0160       LDA *NWLN
0806  8502      0170       STA *OLLN
0808  18        0180       CLC
0809  6980      0190       ADC 80
080B  8504      0200       STA *NWLN
080D  A505      0210       LDA *NWLN+01
080F  6900      0220       ADC 00
0811  C908      0230       CMP 08
0813  D00C      0240       BNE SAME
0815  A504      0250       LDA *NWLN
0817  6927      0260       ADC 27
0819  C952      0270       CMP 52
081B  1008      0280       BPL LAST
081D  8504      0290       STA *NWLN
081F  A904      0300       LDA 04
0821  8505      0310  SAME STA *NWLN+01
0823  18        0320       CLC
0824  60        0330  RTS1 RTS
0825  38        0340  LAST SEC
0826  B0FC      0350       BCS RTS1
                0360  :GENERATE BIRTHS(COLOR=3) & DEATHS(COL=8)
0828  20CA08    0370  GEN0 JSR INIT
082B  200008    0380  GEN1 JSR NXLN
082E  9001      0390       BCC GEN2
                0400  :ALL DONE IF CARRY SET
0830  60        0410       RTS
0831  A027      0420  GEN2 LDY 27
0833  98        0430       TYA
0834  AA        0440       TAX
                0450  :COMP VERT OCC #S
0835  A900      0460  GEN6 LDA 00
0837  994009    0470       STA BUF1,Y
083A  997009    0480       STA BUF2,Y
083D  B102      0490       LDA (OLLN),Y
083F  F00F      0500       BEQ GEN3
0841  1006      0510       BPL GEN7
0843  FE4009    0520       INC BUF1,X
0846  FE7009    0530       INC BUF2,X
0849  2908      0540  GEN7 AND 08
084B  F003      0550       BEQ GEN3
084D  FE4009    0560       INC BUF1,X
```

Note: The stars in the operand indicate zero page mode.

```
0850  B104    0570  GEN3 LDA (NWLN),Y
0852  F00F    0580       BEQ GEN5
0854  1003    0590       BPL GEN4
0856  FE7009  0600       INC BUF2,X
0859  2908    0610  GEN4 AND 08
085B  F006    0620       BEQ GEN5
085D  FE7009  0630       INC BUF2,X
0860  FE4009  0640       INC BUF1,X
0863  88      0650  GEN5 DEY
0864  CA      0660       DEX
0865  10CE    0670       BPL GEN6
0867  A026    0680       LDY 26
0869  18      0690       CLC
086A  AD6709  0700       LDA BUF1+27
086D  6D6609  0710       ADC BUF1+26
0870  8506    0720       STA *SUM1
0872  AD9709  0730       LDA BUF2+27
0875  6D9609  0740       ADC BUF2+26
0878  8507    0750       STA *SUM2
              0760  :COMP OCC #S IN 3X3 & CHANGE COLOR
087A  18      0770  GNLP CLC
087B  A506    0780       LDA *SUM1
087D  793F09  0790       ADC BF1M,Y
0880  38      0800       SEC
0881  F94209  0810       SBC BF1P,Y
0884  8506    0820       STA *SUM1
0886  C903    0830       CMP 03
0888  F00E    0840       BEQ GEN9
088A  9004    0850       BCC GEN8
088C  C904    0860       CMP 04
088E  F00E    0870       BEQ GN10
0890  B102    0880  GEN8 LDA (OLLN),Y
0892  F00A    0890       BEQ GN10
0894  298F    0900       AND 8F
0896  5004    0910       BVC GN16
0898  B102    0920  GEN9 LDA (OLLN),Y
089A  0930    0930       ORA 30
089C  9102    0940  GN16 STA (OLLN),Y
089E  18      0950  GN10 CLC
089F  A507    0960       LDA *SUM2
08A1  796F09  0970       ADC BF2M,Y
08A4  38      0980       SEC
08A5  F97209  0990       SBC BF2P,Y
08A8  8507    1000       STA *SUM2
08AA  C903    1010       CMP 03
08AC  F00E    1020       BEQ GN12
08AE  9004    1030       BCC GN11
08B0  C904    1040       CMP 04
08B2  F00E    1050       BEQ GN13
08B4  B104    1060  GN11 LDA (NWLN),Y
08B6  F00A    1070       BEQ GN13
08B8  29F8    1080       AND 0F8
08BA  5004    1090       BVC GN15
08BC  B104    1100  GN12 LDA (NWLN),Y
08BE  0903    1110       ORA 03
```

```
08C0  9104     1120  GN15 STA (NWLN),Y
08C2  88       1130  GN13 DEY
08C3  F002     1140       BEQ GN14
08C5  10B3     1150       BPL GNLP
08C7  4C2B08   1160  GN14 JMP GEN1
08CA  A904     1170  INIT LDA 04
08CC  8505     1180       STA *NWLN+01
08CE  A900     1190       LDA 00
08D0  8504     1200       STA *NWLN
08D2  8D6809   1210       STA BF1P+26
08D5  8D9809   1220       STA BF2P+26
08D8  60       1230       RTS
               1240  :MOP UP, IF COLOR AND 3 =0, REMOVE(COL=0)
               1250  :OTHERWISE, ALIVE (COL=11)
08D9  20CA08   1260  MOP0 JSR INIT
08DC  200008   1270  MOP1 JSR NXLN
08DF  9001     1280       BCC MOP2
08E1  60       1290       RTS
08E2  A027     1300  MOP2 LDY 27
08E4  B102     1310  MOP3 LDA (OLLN),Y
08E6  F00A     1320       BEQ MOP5
08E8  297F     1330       AND 7F
08EA  C910     1340       CMP 10
08EC  3002     1350       BMI MOP4
08EE  0980     1360       ORA 80
08F0  9102     1370  MOP4 STA (OLLN),Y
08F2  B104     1380  MOP5 LDA (NWLN),Y
08F4  F00A     1390       BEQ MOP7
08F6  29F7     1400       AND 0F7
08F8  6A       1410       ROR
08F9  9002     1420       BCC MOP6
08FB  0904     1430       ORA 04
08FD  2A       1440  MOP6 ROL
08FE  9104     1450       STA (NWLN),Y
0900  88       1460  MOP7 DEY
0901  F0D9     1470       BEQ MOP1
0903  10DF     1480       BPL MOP3
               1490       .EN
```

```
SYMBOL TABLE
OLLN    0002
NWLN    0004
SUM1    0006
SUM2    0007
BUF1    0940
BF1P    0942
BF1M    093F
BUF2    0970
BF2P    0972
BF2M    096F
NXLN    0800
SAME    0821
RTS1    0824
LAST    0825
GEN0    0828
GEN1    082B
GEN2    0831
GEN6    0835
GEN7    0849
GEN3    0850
GEN4    0859
GEN5    0863
GNLP    087A
GEN8    0890
GEN9    0898
GN16    089C
GN10    089E
GN11    08B4
GN12    08BC
GN15    08C0
GN13    08C2
GN14    08C7
INIT    08CA
MOP0    08D9
MOP1    08DC
MOP2    08E2
MOP3    08E4
MOP4    08F0
MOP5    08F2
MOP6    08FD
MOP7    0900
```

```
60230 A=A+1 : C=PEEK(A)-48 : IF C=-16 GOTO 60230
60240 IF C>=0 AND C<9 THEN V=V*10+C : GOTO 60230
60250 S+44 : A=A-1 : RETURN
```

RESEQUENCE can sit quietly behind your program. When you say RUN 60010, your program is renumbered. RESEQUENCE gives error notices if:

A. a GOTO or GOSUB statement wants to go to a non-existant line;
B. there isn't enough room for a new (higher) line number.

In both cases you're given the (new) line number where this happens. RESEQUENCE doesn't run fast (allow about a second per line, more for large programs), but it's dependable and very useful.

Program comments: Line 6000 stops the user program if it gets here. Lines 60010-60020 extract all GOTO, GOSUB, and THEN references and build them into a table. Lines 60030-60040 renumber all lines, and cross-references the table if needed. Line 60050 updates all line references.

Subroutines: 60070 looks for an entry in the line number table. 60090 inserts a new entry into the table. 60110 revises a line number reference. 60160 starts a new scan of the user program; 60170 continues the scan with the next line. 60210 scans the user program for GOTOs, etc.; value S is used to accomodate ON A GOTO ... type situations.

AN APPLE II PAGE 1 MAP

M.R. Connolly Jr.
5009 Rickwood Ct. NW
Huntsville, AL 35810

In the Apple II, the on-screen text is stored in locations $400 through $7FF. Trying to determine just where a particular spot resides in memory isn't easy. The page lines are stored neither consecutively nor sequentially. The APPLE page 1 map shows in hex and decimal the starting and ending locations of each line on the screen. Any given line is sequential from space 1 through space 40; eg, the 20th position of any line is equal to the beginning location +19 decimal or 14 hex.

The value of the page map becomes apparent when used with a listing of the interpretation of numbers stored in the map. Any normal, inverse, or flashing character, or white block, black block, or cursor block may be positioned merely by poking the correct value in the location storing the page position you require.

You might pass this off as just "nice to know" information, but it is very useful if, for instance, you are trying to make an impressive title page for a program you've spent weeks writing. Run the following short program, then try to duplicate it without using the page map and the character chart. It isn't easy!

```
10 CALL -936: FOR I = 1205 TO 1217: POKE I,32: POKE I+ 512,32: NEXT I
20 FOR I = 1333 TO 1589 STEP 128: POKE I,32: POKE I+ 12,32: NEXT I
30 POKE 1463,141: POKE 1465,9: POKE 1467,67: POKE 1469,18: POKE 1471,207
40 GOTO 40
```

MAP OF LINE AND SPACE LOCATIONS FOR TEXT PAGE 1, APPLE II COMPUTER

	LOCATION	
LINE	HEX	DECIMAL
1	400-427	1024-1063
2	480-4A7	1152-1191
3	500-527	1280-1319
4	580-5A7	1408-1447
5	600-627	1536-1575
6	680-6A7	1664-1703
7	700-727	1792-1831
8	780-7A7	1920-1959
9	428-44F	1064-1103
10	4A8-4CF	1192-1231
11	528-54F	1320-1359
12	5A8-5CF	1448-1487
13	628-64F	1576-1615
14	6A8-6CF	1704-1743
15	728-74F	1832-1871
16	7A8-7CF	1960-1999

17	450-477	1104-1143
18	4D0-4F7	1232-1271
19	550-577	1360-1399
20	5D0-5F7	1488-1527
21	650-677	1616-1655
22	6D0-6F7	1744-1783
23	750-777	1872-1911
24	7D0-7F7	2000-2039

Not used for on-screen display: 478-47F; 4F8-4FF; 578-57F; 5F8-5FF; 678-67F; 6F8-6FF; 778-77F; 7F8-7FF

MACHINE INTERPRETATION OF VALUES STORED IN $400.7FF APPLE II COMPUTER

FIGURE	NORMAL	INVERSE	FLASH	FIGURE	NORMAL	INVERSE	FLASH
@	128,192	0	64	!	161,225	33	97
A	129,193	1	65	"	162,226	34	98
B	130,194	2	66	#	163,227	35	99
C	131,195	3	67	$	164,228	36	100
D	132,196	4	68	%	165,229	37	101
E	133,197	5	69	&	166,230	38	102
F	134,198	6	70	'	167,231	39	103
G	135,199	7	71	(	168,232	40	104
H	136,200	8	72	)	169,233	41	105
I	137,201	9	73	*	170,234	42	106
J	138,202	10	74	+	171,235	43	107
K	139,203	11	75	,	172,236	44	108
L	140,204	12	76	-	173,237	45	109
M	141,205	13	77	.	174,238	46	110
N	142,206	14	78	/	175,239	47	111
O	143,207	15	79	Ø	176,240	48	112
P	144,208	16	80	1	177,241	49	113
Q	145,209	17	81	2	178,242	50	114
R	146,210	18	82	3	179,243	51	115
S	147,211	19	83	4	180,244	52	116
T	148,212	20	84	5	181,245	53	117
U	149,213	21	85	6	182,246	54	118
V	150,214	22	86	7	183,247	55	119
W	151,215	23	87	8	184,248	56	120
X	152,216	24	88	9	185,249	57	121
Y	153,217	25	89	:	186,250	58	122
Z	154,218	26	90	;	187,251	59	123
[	155,219	27	91	<	188,252	60	124
\	156,220	28	92	=	189,253	61	125
]	157,221	29	93	>	190,254	62	126
^	158,222	30	94	?	191,255	63	127
_	159,223	31	95				
(BLOCK)	160,224	32	96				

EXPLORING THE APPLE II DOS

Andy Hertzfeld
2511 Hearst St. Apt. 204
Berkeley, CA 94709

To say that the documentation which comes with Apple's Disk II system is skimpy is being very kind. Only a terse description of each DOS command is provided and absolutely zilch is said about its memory usage or internal structure. Hopefully, Apple will soon remedy this situation but until that time hobbyists must rely on each other for the vital information. I have been exploring the internals of the DOS for the last few months; this article summarizes some of the interesting things I've found.

The DOS resides in the highest portion of your system's memory and is about 10K bytes long. Its exact size depends on how many file buffers you choose to allocate (one file buffer is needed for each simultaneously open file). Each file buffer is 595 bytes long and the system provides you with three to start with (you must have at least one).

The DOS communicates with the rest of the system via the input and output hooks CSW and KSW located at $36 - $39 (This article uses "$" to indicate a hexadecimal number). Through these hooks it is given control every time a character is inputted or outputted. This is a nice scheme because it allows the DOS to be called from any environment (BASIC, Monitor, Mini-Assembler, etc.) but it has the drawback of activating the DOS when a command is typed as input to a user program, which is usually not what you want. Also, since the reset button resets the hooks, the DOS is disabled whenever the system is reset, which isn't so great.

The process of loading the DOS into memory for the first time is called "bootstrapping." Bootstrapping is initiated when control is transferred to the PROM on the disk controller card. Memory pages 3 and 8 are blown by a bootstrap. There are two different types of disks you can boot from: masters and slaves. The distinction is that a master disk can be used to bootstrap on a system of arbitrary memory size while a slave will only work properly on a system with the same memory size as that which created it. This is because since the DOS sits at the top of memory, its addresses (for JSRs, JMPs, etc.) will be different on systems with different memory sizes. A master disk cleverly solves this problem by loading into low memory first and then relocating itself up to where it belongs. Note that this means that a master bootstrap will blow alot of additional memory.

All addresses in this article are for a 48K system. If your system has memory size X, subtract 48K - X from the addresses that are given here.

A call to the routine at $9DB9 will initialize or re-initialize the DOS. This routine should be called after every reset to restore the hooks. It is exactly like typing "3DO" "G" as Apple's documentation recommends but is a little bit safer since the $3DO location is often destroyed by various programs.

Every diskette has a volume number from 1 to 254 associated with it. It is assigned when the diskette is initialized and there is currently no easy way to change it. The volume number of the current disk is stored at $B7F6. Before most DOS commands the system checks to see if the current volume number matches the last volume number used. If it doesn't, a "volume mismatch" error is generated. While this "feature" may be nice for large business applications that don't want dumb operators inserting the wrong disks, it is very annoying to most average users, especially when you want to transfer a number of programs between two disks with different volume numbers. After much searching, I located the place where the volume check is performed and devised a patch to disable it. It's only two bytes long; just enter the monitor and type: "BDFE: A9 00 ". This will disable all volume checking until the next boostrap. It works by replacing the comparison instruction which performs the volume check with a " LDA #0 " instruction which sets the "equality" or Z flag, effectively forcing the match to succeed.

Binary files of arbitrary length can be saved on disk with the "BSAVE" command. Each BSAVEd file has an implicit starting address and length associated with it; when the file is BLOADed it is loaded at the starting address. Unfortunately, there is no way provided for a user to find out the starting address and length of a BSAVEd file; this makes copying files that you are not intimately familiar with very difficult.

Fortunately, when a file is BLOADed, the directory record of the file is always placed in a buffer in a fixed location. The buffer contains the starting address and length of the file as well as other useful information. The length is kept at memory locations $A9A3 - $A9A4 while the starting address is stored at $A9B5 - $A9B6 (with the least significant byte first, as usual). Thus to retrieve the starting address and length of a BSAVEd program you can simply BLOAD it and then peek at the above locations.

Some people might wish to alter the names of some of the DOS commands to suit their own, personal tastes (it is, after all, a personal computer). For example, I know many folks would like to abbreviate the "CATALOG" command to a simple "C". This is surprisingly easy to do; since the DOS lives in RAM the contents of its command table are easily changed. The command table is located from $A7E0 - $A863. Each command name is represented as an ASCII string with the high bits off, except for the last character of the string, which has its high-order bit set. The strings are associated with the commands by their position in the command table (the first string corresponds to the INIT command, the second to the LOAD command, etc.). The position of every command is given below in Table 1.

Thus you can dream up your own names for the commands by storing new strings in the command table. For example to change the name of the INIT command to "DNEW" you would enter the monitor and type " A7E0: 44 4E 45 D7". However, some caution is required when you change the length of a command name; in general you will probably have to rewrite the entire command table to achieve the desired affect.

The error message table is stored at addresses $A8CD - $A980. By using the same techniques described for the command table, you can rewrite the error messages to be whatever you like.

TABLE 1: POSITION OF COMMANDS IN THE COMMAND TABLE

The position refers to which string in the command table is associated with the command. 1 means its the first string, etc.

Position	Command
1	INIT
2	LOAD
3	SAVE
4	RUN
5	CHAIN
6	DELETE
7	LOCK
8	UNLOCK
9	CLOSE
10	READ
11	EXEC
12	WRITE
13	POSITION
14	OPEN
15	APPEND
16	RENAME
17	CATALOG
18	MON
19	NOMON
20	PR#
21	IN#
22	MAXEILES
23	EP
24	INT
25	BSAVE
26	BLOAD
27	BRUN
28	VERIFY

It is hard to use the input and output hooks in conjunction with the DOS since you cannot simply change the hooks as they are the DOS' only contact with the rest of the system. Also, if you only change one of them, the DOS has the nasty habit of changing it back. Fortunately, the DOS has its own internal hooks it uses for keyboard input and video output. Its output hook is at $A996 - $A997 and the input hook immediately follows at $A998 $A999. If you change the contents of these addresses instead of the usual hooks at $36 - $39, everything should work just fine. Eor example, lets say you wanted to divert output to a line printer without disabling the DOS. If the line printer output routine is located at $300, all we would have to do is enter the monitor and type " A996: 00 03 ".

To execute a DOS command from a BASIC program, you simply print it, prefixing it with a "control-D". The prefix character is stored at memory location $A9E5, with its high-order bit set. Thus, if you don't like control-D and wish to use some other prefix character, all you have to do is store a different character value into $A9E5.

I am very curious to find out the primitive instructions the DOS uses to communicate with the disk controller, but without proper documentation it is very difficult to determine what does what (Can someone out there help me?). I have managed to find out the primitives that turn the drive on and off, though. If your controller card is in slot S, referencing memory location $C089 + $SO will power up the disk and start it spinning while referencing $C088 + $SO will turn it back off.

This article is merely the tip of the proverbial iceburg; most of the DOS's internals still remain a mystery to me. I hope Apple eventually distributes complete documentation but until then other curious users can use this article as a starting point for their own explorations and hopefully report back what they find. Table 2 (below) contains a summary of important addresses in the DOS for easy reference, including some not mentioned in the above commentary.

TABLE 2: IMPORTANT ADDRESSES IN THE APPLE II DOS

Address	Function
$B7F6	holds the volume number of the current diskette
$9DB9	routine to re-initialize the DOS
$A9E5	location of printing command character, initially set to control-D
$A9B5 - $A9B6	starting address of most recently loaded program, lsb first
$A9A3 - $A9A4	length of most recently loaded program
$A7E0 - $A863	the DOS command table
$A8CD - $A980	the DOS error message table
$A996 - $A997	the internal hook address to output a character
$A998 - $A999	the internal hook address to input a character
$C089 + $S0, S= slot no.*	address to power up the disk
$C088 + $S0, S= slot no.*	address to power down the disk
$9E4D	routine which handles the input hook
$9E7E	routine which handles the output hook
$BD00	routine which reads in the directory off the disk. It is called by virtually every DOS command

All addresses given (except those marked with an asterisk) refer to a system with 48K bytes of memory. If your system has memory size X, subtract (48K-X) from each address.

HOW DOES 16 GET YOU 10?

Gary P. Sandberg
1144 Amber Ridge Drive
Lilburn, GA 30247

In order to PEEK, POKE, figure CALL numbers, etc. effectively a knowledge of Hexadecimal / Decimal conversion is a necessity. My experience during the past ten years, working with computer systems and data processing equipment did not include anything that required hexadecimal addressing and coding. When I started using my Apple II, I was completely lost and confused with base 16 math. I began looking for a way to work with hexadecimal effectively. The following conversion table was the answer.

HEXADECIMAL / DECIMAL CONVERSION TABLE

	16^3	16^2	16^1	16^0
0	0	0	0	0
1	4,096	256	16	1
2	8,192	512	32	2
3	12,288	768	48	3
4	16,384	1,024	64	4
5	20,480	1,280	80	5
6	24,576	1,536	96	6
7	28,672	1,792	112	7
8	32,768	2,048	128	8
9	36,864	2,304	144	9
A	40,960	2,560	160	10
B	45,056	2,816	176	11
C	49,152	3,072	192	12
D	53,248	3,328	208	13
E	57,344	3,584	224	14
F	61,440	3,840	240	15

To convert a number from hexadecimal to decimal;

1. in each column of the table, find the decimal equivalent for the hexadecimal digit in that position.
2. add the decimal equivalents, found in step #1, to obtain the decimal number.

Hopefully the following examples will help you master the use of the conversion table.

To convert a number from decimal to hexadecimal;

1. In the table find the largest decimal value that will fit into the decimal number to be converted.
2. note its **column position** and hexadecimal equivalent.
3. find the decimal remainder (subtract)
4. repeat steps 1, 2, & 3 for each remainder. When a hexadecimal equivalent has been found in the **right most column**, the conversion is done.

Convert Hex to Decimal using the conversion table.

16^3	16^2	16^1	16^0			Convert from left to right.
			E_{16}	=	14	
		5		=	80	list and
	E			=	3584	ADD TOGETHER
F				=	61440	
			$FE5E_{16}$	=	65118_{10}	

Convert Decimal to Hex using the conversion table.

		16^3	16^2	16^1	16^0
65118					
-61440	from table =	F			
3678					
-3584	from table =		E		
94					
-80	from table =			5	
14	from table =				E

65118_{10} = $FE5E_{16}$

Remember the Apple II's system monitor can help you with some of your hexadecimal problems. The monitor will do hexadecimal addition and subtraction, as shown on page 70 of the Apple II reference manual.

The Apple II's PEEK function also can be helpful. In BASIC key in PRINT PEEK (2), the Apple II will display on the screen the decimal value of decimal memory location 2.

Use the POKE statement to change memory location 2, In BASIC key in POKE 2,255, then Return. Then PRINT PEEK (2), Return. The Apple will display 255.

Then CALL -151, or hit Reset. The Apple II is now in the System Monitor. Key in 0002 or 2, Return, and the Apple II displays 0002-FF. Why?, because we put the decimal value 255 into memory location 2 with the POKE statement, 255(10) is equal to FF(16), get the idea?

For some conversions from hexadecimal to decimal or back the other way, you can use the POKE and PEEK method, but for most conversions use the table.

Here are two more examples that don't use a conversion table: same numbers different method of conversion:

Convert Hex to Decimal without using the conversion table.

First digit is	* 1	E * 1	= 14
Second digit is	* 16	5 * 16	= 80
Third digit is	* 256	E * 256	= 3584
Fourth digit is	* 4096	F * 4096	= 61440

$FE5E_{16}$ = 65118_{10}

Convert Decimal to Hex without using the conversion table.

65118 / 16 =	4069.875 ->	.875 * 16 = 14	= E
4069 / 16 =	254.3125 ->	.3125 * 16 = 5	= 5
254 / 16 =	15.875	.875 * 16 = 14	= E
15 / 16 =	.9375	.9375 * 16 = 15	= F

65118_{10} = $FE5E_{16}$

Use either method to convert from one number system to the other, and with a little practice you will be converting numbers with speed and accuracy.

APPLE II - TRACE LIST UTILITY

Alan G. Hill
12092 Deerhorn Rd.
Cincinnati, OH 45240

Did you ever use the TRACE function in Integer BASIC, only to give up in despair after looking at a screeen full of line numbers? Try it without a printer and you may never use TRACE again! Well, here's the utility that will put TRACE back into your debugging repertoire (for those of us who need a little help getting it right.)

The utility presented here will list each BASIC program source statement line by line in the order executed. There's no need to refer back and forth between TRACE line numbers and the source program listing. Two versions are presented: Version 1 is a real-time utility; i.e. each statement is listed immediately prior to execution so you can follow the programs logical sequence. You can slow the execution rate down or even temporarily halt execution while you scan the screen. Version 2 only saves the line numbers of the last 100 lines executed for listing later. Version 3 could be useful in tracing a full-screen graphics program.

The Technique

The program utilizes the COUT hook at $36.37 to intercept and suppress TRACE printing. All other printing continues normally with one exception (see Warning #1). Before returning to the BASIC interpreter, the line number is picked up and pushed into an array (TR) in the variables area above LOMEM. If the number is the same as the previous line number, a zero line number is placed in the stack with the line number of a FOR I = 1 to 1000: NEXT I delay loop, for instance. When the number changes, it will be placed in the stack. The most recent 100 line numbers are saved. Tracing is performed under user control by the normal TRACE/NOTRACE statements. In Version 2, the lines may then be listed after the test program ends. The technique in Version 1 is similar with one distinction. The trace intercept routine transfers control to the utility program to list the line as soon as it is put in the stack.

How The TRACE Intercept Routine Works

The output pointer in $36.37 is initialized by the utility to the address ($300) of the Trace Intercept Routine. Each character is examined by TIR as it comes through if the TRACE flag is up (bit 7 of $AO on). If off, TIR jumps back to the normal print utility at $FDFO. If the character is a # ($A3), it is assumed that a line number follows. Every line number in the stack is pushed down and the current line number is placed at the top. Location $DC.DD points to the BASIC line about to be executed. The line number is in the second and third bytes. In Version 2, TIR returns to the interpreter. In the real-time version (Version 1), control is next transferred to the utility program at line 30020. TIR expects that the address of line 30010 has been saved in $15.16 by the utility programs CALL 945 in line 30010. TIR first saves the contents of $DC.DD and then replaces it with the contents of $15.16. It also saves the address of the current statement within the BASIC line. That is, the contents of $EO.E1 are saved at $1B.1C. TIR can now transfer control back to the interpreters continue entry point by a JMP $F88A which then executes line 30020 of the utility. The current line of the test program is listed; the BASIC pointers are restored by the CALL 954 in line 30090; the return address is popped; and control is returned to the test program through $E881. Fait accompli.

As mentioned previously, the TR array is used to save the line numbers. The array is set up the first time TIR is entered. Note that TR is intentionally not DIMensioned in the utility. TIR must handle that task since a RUN of the test program will reset the variables area pointer ($CC.CD) back to LOMEM.

Programming The Routines

TIR starts at $300. It could be relocated if the absolute references in the POKE and CALL statements are changed. Also note that the LIST statement in lines 30060 and 32040 will not be accepted by the Syntax checker. They must first be coded as PRINT statements, located, and changed to LIST tokens ($74) using the monitor. This is more easily done if these lines are coded and the tokens changed before the remaining lines are entered. See example below for the case where HIMEM is 32768:

```
NEW
30060 PRINT EXECLINE
32040 PRINT TR (I)
(hit reset to enter monitor)
*7FEC;74
*7FF9:74
(enter Control/C)
LIST
30060 LIST EXECLINE
32040 LIST TR (I)
```

Using The Utility

1. After coding the assembler and BASIC utility programs, the test program is then appended. This may be done by a RUN 31000. Start the cassette recorder and hit Return when prompted. The test program will be appended to the utility program provided its highest line number is less than 29970.

2. Create a line O that will be used to indicate that a line has successively executed. For example, code:
   O REM ***ABOVE LINE REPEATED***

3. **Run** the utility of your choice:
   RUN 30000 Version 1 (Real-time list)
   or RUN 32000 Version 2 (Post-execution list)

4. Insert the TRACE/NOTRACE statements wherever desired in test program. Just enter the TRACE command directly if you want to trace the entire program. Also see Warning #1.

5. RUN the test program.

6. Display the results:
   A. **Real-time Version:** The lines will be listed automatically as executed. Note the FOR: NEXT loop in line 30090 can be adjusted to control the execution rate. The upper limit could be PDL(O), thereby giving you run-time control over the execution rate. Note also that execution can be forced to pause by depressing paddle switch O. Execution will resume when the switch is released.

B. **Post-execution Version:** After stopping or ending the program, enter a GOTO 32020 command. The first page of statements will be displayed. Enter a "C" to display additional pages, a "T" to reset for another test run, or an "E" to return to BASIC. Note that even if you have traced with Version I, you can still display the last 100 lines with Version 2.

Sample Run

Test Program

```
0     REM *** REPEATED ***
10    TRACE
30    GOSUB 100+RND(3) *10
40    FOR I=1 TO 10: NEXT I
50    GOTO 30
100   PRINT "LINE 100":RETURN
110   PRINT "LINE 110":RETURN
120   PRINT "LINE 120":POP
125   NO TRACE:END

>     RUN 30000
>     RUN
```

Trace Output

```
    30   GOSUB 100+RND(3)*10
    110  PRINT "LINE 110":RETURN
LINE 110
    30   GOSUB 100+RND(3)*10
    40   FOR I=1 TO 10:NEXT I
    0    REM *** REPEATED ***
    50   GOTO 30
    30   GOSUB 100+RND(3)*10
    120  PRINT "LINE 120":POP
LINE 120
    125  NO TRACE:END

    >
```

For a slow motion game of "BREAKOUT", trace it with the real-time version!

Hints And Warnings

It's usually a good idea to deactivate TIR after the test program has ended by hitting Reset and Control/C and entering NOTRACE. Don't try to trace the test program without first running the utility program at line 30000 or 32000.

To increase the debugging power of the real-time trace utility, make liberal use of the push button to halt program execution. With practice and the proper choice of the delay loop limit in line 30090, you can step through the program one line at a time. Enter a Control/C while the push button is depressed and execution will be STOPPED AT 30070. You can then use the direct BASIC commands to PRINT and change the current value of the programs variables. Enter CON and execution will resume.

With additional logic in the utility program, you can create specialized tracing such as stopping after a specified sequence of statements has been detected. Return via a CALL 958 if you don't want TRACE turned back on.

Tracing understandably shows the execution rate of your program, but you probably aren't concerned with speed at this point. However, the wise use of TRACE/NOTRACE will help move things along. Also, when encountering a delay loop such as FOR I=1 to 3000: NEXT I, you may want to help it along by stopping with a Control/C entering I=2999, and CONtinuing.

Warning #1: There must be **no** PRINT statement with a # character in the output. TIR assumes that a # is the beginning of a trace sequence. Either remove the # or bracket the PRINT statement with a NOTRACE/TRACE pair.

Warning #2: There must be no variable names in the test program identical to those in Version 1. The TR variable name must be unique in both versions.

Warning # 3: Line O in the test program should be a REMark statement as described above to avoid confusion. Line O is listed when a line is successively repeated.

Warning # 4: Once TRACE has been enabled, the test program must not dynamically reset the variables pointer ($CC.CD) with a CLR or POKE unless it first disables TRACE and resets $13.14; e.g., 100 NOTRACE:CLR: POKE 19, O: POKE 20,0: TRACE is OK.

Extensions

The primary motivation for this program was to improve the TRACE function in Integer BASIC. However, one can imagine other uses of a program that gains control as each statement is executed — maybe the kernel of a multiprogramming executive. I would be interested in seeing your comments and modifications.

ZERO PAGE MEMORY MAP

Location	Use
$00.01	SAVE AREA FOR HIMEM. APPEND USES
$05	PROGRAM SWITCH ON=$FF OFF=$7F Turned on when trace # character ($a3) is detected. Turned off when next space character ($A0) is detected
$13.14	ADDRESS OF TR STORAGE VARIABLE
$15.16	ADDRESS THAT CAUSES RETURN TO LINE 30020 IN BASIC LIST UTILITY (Version I)
$17.18	SAVE AREA FOR $DC.DD. ADDRESS OF CURRENT BASIC LINE IN TEST PROGRAM
$1B.1C	SAVE AREA FOR $EO.E1. ADDRESS OF STATEMENT WITHIN BASIC LINE
$A0	APPLE II TRACE FLAG ON=$FF OFF=$7F

```
                TRACE INTERRUPT ROUTINE

                BY ALAN G. HILL
                23 NOVEMBER 1978

                COMMERCIAL RIGHTS RESERVED

0300                    ORG   $030C

0300 24 A0      START   BIT   $00A0  IS TRACE ON?
0302 30 03              BMI   $0307  BRANCH YES
0304 4C FD FD   PRINT   JMP   $FDF0  NO. BACK TO PRINT
0307 C9 A3      TRACE   CMPIM $A3    NUMBER SIGN?
0309 F0 CD              BEQ   SWON   BRANCH YES. IT'S A TRACE LINE
030B 24 05              BIT   $0005  SWITCH ON?
030D 10 F5              BPL   PRINT  NO. PRINT CHARACTER
030F C9 A0              CMPIM $AC    SPACE?
0311 D0 04              BNE   RETURN NO. RETURN W/O PRINTING
0313 4C D3 03           JMP   TRCOFF VER. II LDAIM $7F
0316 EA                 NOP          VER. II STA $05
0317 60         RETURN  RTS          BACK TO BASIC

0318 A9 FF      SWON    LDAIM $FF    TURN ON SWITCH
031A 85 05              STA   $0005
031C A5 13              LDA   $0013  FIRST TIME THRU?
031E D0 49              BNE   GLINO  BRANCH NO. TO GET LINE NO.
0320 A5 14              LDA   $0C14
0322 D0 45              BNE   GLINO
0324 A5 CD              LDA   $00CD  YES. SETUP TR ARRAY
0326 85 14              STA   $0014  IN VARIABLES
0328 A5 CC              LDA   $00CC  AREA AND ADJUST
032A 85 13              STA   $0013  POINTER
032C 18                 CLC
032D 69 CF              ADCIM $CF
032F 85 CC              STA   $00CC  NEW PV
0331 A5 CD              LDA   $00CD
0333 69 00              ADCIM $00
0335 85 CD              STA   $00CD
0337 A0 00              LDYIM $00
0339 A9 D4              LDAIM $D4    "T"
033B 91 13              STAIY $13
033D C8                 INY
033E A9 D2              LDAIM $D2    "R"
0340 91 13              STAIY $13
0342 C8                 INY
0343 A9 00              LDAIM $00
0345 91 13              STAIY $13    DSP
0347 C8                 INY
0348 A5 CC              LDA   $00CC
034A 91 13              STAIY $13    NVA
034C C8                 INY
034D A5 CD              LDA   $00CD
034F 91 13              STAIY $13
0351 18                 CLC
0352 A9 04              LDAIM $04    POINT $13.14 TO TR
0354 65 13              ADC   $0013  DATA AREA-1
```

```
0356 85 13           STA   $0013
0358 A5 14           LDA   $0014
035A 69 00           ADCIM $00
035C 85 14           STA   $0014
035E A0 CA           LDYIM $CA     INITIALIZE TR ARRAY
0360 A9 FF           LDAIM $FF     TO ALL FF'S
0362 91 13     FLOOP STAIY $13
0364 88              DEY
0365 D0 FB           BNE   FLOOP   LOOP TIL DONE
0367 F0 29           BEQ   SLINE   ALWAYS

0369 A0 02     GLINO LDYIM $02
036B B1 13     TLINE LDAIY $13     IS LAST LINE NO.
036D D1 DC           CMPIY $DC     SAME AS THIS ONE?
036F D0 08           BNE   NLINE   BRANCH NO
0371 88              DEY
0372 D0 F7           BNE   TLINE   LOOP
0374 98              TYA           YES. PUT ZERO
0375 48              PHA           LINE NO. IN
0376 48              PHA           STACK TEMPORARILY
0377 F0 21           BEQ   TSTACK ALWAYS

0379 A0 02     NLINE LDYIM $02     IS THERE ALREADY A
037B B1 13     TLOOP LDAIY $13     ZERO AT THE TOP?
037D D0 13           BNE   SLINE   BRANCH NO TO GET LINE NO.
037F 88              DEY
0380 D0 F9           BNE   TLOOP   LOOP
0382 A2 02           LDXIM $02     YES
0384 C8              INY
0385 B1 DC     CLOOP LDAIY $DC     COMPARE WITH NEXT
0387 C8              INY           LAST LINE NO.
0388 C8              INY
0389 D1 13           CMPIY $13
038B D0 05           BNE   SLINE   IT'S DIFFERENT. SAVE IT
038D 88              DEY           IT'S SAME
038E CA              DEX
038F D0 F4           BNE   CLOOP   LOOP
0391 60              RTS           STILL THE SAME.  RETURN TO TRACE

0392 A0 02     SLINE LDYIM $02
0394 B1 DC     PLINE LDAIY $DC     PICK UP LINE NO.
0396 48              PHA           HOLD IN STACK TEMPORARILY
0397 88              DEY
0398 D0 FA           BNE   PLINE   BOTH DIGITS
039A A0 CB     TSTACK LDYIM $CB    PUSH DOWN ALL TR
039C B1 13     PLOOP LDAIY $13     ELEMENTS TO
039E C8              INY           MAKE ROOM FOR
039F C8              INY           NEW LINE NO. AT TR(0)
03A0 91 13           STAIY $13
03A2 88              DEY
```

```
03A3 88              DEY
03A4 88              DEY
03A5 D0 F5           BNE    PLOOP  LOOP UNTIL DONE
03A7 A0 01           LDYIM $01
03A9 68              PLA           PUT NEW LINE NO. OR
03AA 91 13           STAIY $13     ZERO IN TR(0)


03AC C8              INY
03AD 68              PLA           GET HIGH ORDER BYTE
03AE 91 13           STAIY $13     STUFF IT TOO
03B0 60              RTS           RETURN TO BASIC

03B1 A5 DC     SAVE  LDA    $00DC  ROUTINE TO SAVE ADDRESS
03B3 85 15           STA    $0015  SO TIR WILL CAUSE BASIC
03B5 A5 DD           LDA    $00DD  TO EXECUTE LINE 30020
03B7 85 16           STA    $0016  WHEN TRACE SEQUENCE IS DETECTED
03B9 60              RTS           RETURN TO UTILITY

03BA A9 FF     TEST  LDAIM $FF     ROUTINE TO RE-ENTER TEST PGM
03BC 85 A0           STA    $00A0  TURN TRACE BACK ON
03BE A5 17           LDA    $0017  RESTORE TEST PROGRAM
03C0 85 DC           STA    $00DC  LINE NO.
03C2 A5 18           LDA    $0018
03C4 85 DD           STA    $00DD  AND
03C6 A5 1B           LDA    $001B
03C8 85 E0           STA    $00E0  STATEMENT ADDRESS
03CA A5 1C           LDA    $001C
03CC 85 E1           STA    $00E1
03CE 68              PLA           POP UTILITY ADDRESS
03CF 68              PLA           FROM STACK
03D0 4C 81 E8        JMP    $E881  RE-ENTER BASIC TRACE ROUTINE

03D3 A9 7F     TRCOFF LDAIM $7F    TURN OFF
03D5 85 05           STA    $0005  SWITCH AND
03D7 85 A0           STA    $00A0  TRACE: (DON'T TRACE UTILITY)
03D9 A5 DC           LDA    $00DC
03DB 85 17           STA    $0017  SAVE ADDRESS OF
03DD A5 DD           LDA    $00DD  TEST PGM LINE NO.
03DF 85 18           STA    $0018
03E1 A5 15           LDA    $0015  SETUP TO TO TO UTILITY
03E3 85 DC           STA    $00DC  TO LIST LINE NO.
03E5 A5 16           LDA    $0016  SETUP LINE ADDRESS
03E7 85 DD           STA    $00DD
03E9 A5 E0           LDA    $00E0  SETUP STATEMENT ADDRESS
03EB 85 1B           STA    $001B
03ED A5 E1           LDA    $00E1
03EF 85 1C           STA    $001C
03F1 68              PLA           REMOVE ADDRESS FROM STACK
03F2 68              PLA
03F3 4C 8A E8        JMP    $E88A  GO TO UTILITY VIA CONTINUE
```

VERSION I: Real-Time Trace List Utility Program

```
29770  REM REAL-TIME TRACE LIST UTILITY PROGRAM
29980  REM SET-UP COUT AND INITIALIZE ZERO PAGE VALUES
29990  REM SET-UP TIR ASSEMBLER JUMP
30000  NOTRACE; POKE 54,768 MOD 256: POKE 55,768/256:
       POKE 19,0:POKE20,0=POKE 787,76: POKE 788,211:
       POKE 789,3: POKE 790,234
30005  REM SAVE ADDRESS SO TIR RETURNS TO LINE 30020
30010  CALL 945 :END
30020  EXECLINE=TR(0): IF EXECLINE #0 THEN 30050
30030  IF RRRRR=1 THEN 30070
30040  RRRRR=1: GOTO 30060
30050  RRRRR=0
30060  LIST EXECLINE
30070  IF PEEK (-16287)>127 THEN 30070: REM PAUSE IF SW(0) ON
30080  IF EXECLINE = 0 THEN 30100: REM SKIP DELAY
30090  FOR JJJJJ=1 TO 100: NEXT JJJJJ: REM DELAY
30100  CALL 954: REM BACK TO TEST PGM
30110  END: REM NEVER EXECUTED

31000  REM APPEND TEST PROGRAM
31010  INPUT "HIT RETURN TO APPEND" A$
31020  POKE 0, PEEK(76): POKE 1, PEEK (77): POKE 76, PEEK (202):
       POKE 77, PEEK (203): CALL-3873: POKE 76, PEEK (0):
       POKE 77, PEEK (1):END
```

VERSION II: Post-Execution Trace List Utility Program

```
32000  NOTRACE: POKE 54,768 MOD 256: POKE 55,768/256: POKE 19,0:
       POKE 20,0: POKE 787,169: POKE 788,127:
       POKE 789,133: POKE 790,5
32010  PRINT "TRACE SET UP. ENABLE TRACE IN TEST PROGRAM": END
32015  REM GOTO 32020 WHEN TEST PGM ENDED
32020  NOTRACE: POKE 54,240: POKE 55,253:
       IF PEEK (20)#0 THEN 32030: PRINT "TRACE
       WAS NOT ON IN TEST PROGRAM": GOTO 32090
32030  CALL-936: FOR I=100 TO 1 STEP-1:
       IF TR (I)=-1 THEN 32060
32040  LIST TR (I)
32050  IF PEEK (37)>18 THEN 32090
32060  NEXT I
32070  GOTO 32090
32080  CALL-936: IF I>1 THEN 32060
32090  PRINT:PRINT "C/T/E?"
32100  KEY=PEEK(-16384): IF KEY< 128 THEN 32100:
       POKE-16368,0: IF KEY=212 THEN 32000:
       IF KEY=195 THEN 32080:END
```

6522 CHIP SETUP TIME

John T. Kosinski
4 Crestview Drive
Millis, MA 02054

Richard F. Suitor
166 Tremont Street
Newton, MA 02158

MICRO 6:4 summarized some discussion from EDN concerning their difficulties with interface design. One point in particular caught our eye - a statement that the 6522 VIA chip cannot use the Apple-generated device select signal (from pin 41 of the I/O slot) because the data sheets clearly require that the chip be selected 180 ns before the I/O enable signal goes high, whereas the Apple-generated signals occur nearly simultaneously. That is a misconception which we would like to correct. We report a 6522 interface that uses the pin 41 select signal, that theoretically ought to work and in fact does work.

The 6522 VIA - Why Bother?

Since there are several interfaces both supplied by Apple and by other vendors, why bother? VIA stands for Versatile Interface Adapter. It was designed by MOS Technologies, the same folks who brought us the 6502 and it is well named. It has two I/O ports, two timers and a shift register, and so many options in operating them that we won't try to list them. A very useful feature is that all of the functions can interrupt the 6502. Several software tasks (cassette I/O, music, software generated serial I/O) require the Apple to spend most of its time in timing loops. With the use of timers and interrupts, these functions can be performed while the system is running some other program. You can have your STAR WARS theme while shooting TIE fighters, instead of after; more prosaically, you can print edited text while editing more. The 6522 is quite flexible because of its versatility; it is a definite asset to the Apple.

What's the Big Problem?

The 6522 was designed to work well with the 6502. The signals at the Apple I/O slots are not all 6502 signals, however - some are decoded device select signals, which would be very convenient to use if we could. According to the referenced letter, we can't - there is not enough time to select the chip. As mentioned before, the problem is not insurmountable; let's discuss timing a bit. The 6522 has 16 registers that control all the bells and whistles. To communicate with the 6522 from the CPU, one:

1. Selects one of the 16 registers with the address lines.
2. Selects (turns on) the 6522 chip itself.
3. Enables the I/O transaction.
4. Disables the I/O transaction.
5. De-selects the chip.

Some of the processes take time. For example, th 6522 data sheets DO say that the address must be valid 180 ns before the I/O enable. They ALSO state that the select is normally derived from the address lines. However, the timing tolerance referred to is the register select operation of step 1, and it must occur 180 ns before the I/O enable of step 3. The data sheets DO NOT specify the chip select time of Step 2. A representative of MOS Technologies, looking at the circuit diagrams, estimated that it would be sufficient to have Step 2 occur 40 - 50 ns before Step 3. He did not offer a minimum lead time requirement.

The 6502 and the 6522 expect that Step 3 will occur when the 6502 02 signal goes high and that Step 4 will occur when 02 goes low. The enable signal presented at the I/O port of the Apple is actually $\overline{\phi 0}$, a signal which leads 02 by 50 - 70 ns. That is a very short time, but long compared to the 10 ns or so it takes an LS gate to operate. There are three LS gates involved in a transfer (the chip itself, and data bus buffers at each end) giving a nominal 30 ns timing tolerance. Actually, if the devices on the data bus are properly tristated (i.e. they have very high impedance unless they are active), the capacitance of the bus and the buffer delays will probably permit proper operation with the $\overline{\phi 0}$ enable pulse. There certainly seem to be several circuits using that signal that work (now including, for some unknown reason, EDN's.)

In summary, there are perhaps two problems in interfacing a 6522 to the Apple:

1. One may indeed need to select the chip before enabling the I/O, but no more than 40 - 50 ns before.
2. One may need to use an I/O enable signal that is coincident (within about 30 ns) with the 6502 02.

It is not at all clear what one could get away with if one tried; it is clear that if the requirements 1 and 2 are met, the 6522 should interface easily to the Apple II. However, since the device select and I/O select signals that Apple supplies de-select at the end of $\overline{\phi 0}$, one should reasonably expect that an interface that tristates when these signals deselect should work satisfactorily with the Apple despite the fact that the 6502 is accepting data for another 50 ns. It is apparent from the discussion that has resulted from EDN's efforts that many interfaces so designed do work satisfactorily; it is not clear how marginal the operation is.

There is an interesting discussion of the Apple timing in the Sept. issue of KILOBAUD starting on page 10. They reported on a 6522 interface and found that the important time was the rise of the I/O enable signal. Since they do not mention what was done for chip select and for data bus buffering, one can only wonder if chip select timing was affecting their results.

We decided to play safe and satisfy both requirements. One way to satisfy the second is to use the real 02. As it turns out, this also satisfies the first, because 02 lags the device select signal by about 50 ns. This coincidence may have led to some confusion in interpreting timing experiments! This is the approach we followed; in retrospect, knowing what we do now, we would have proceded otherwise (i.e. perhaps used a delayed device select signal as an I/O enable signal.) Since it does no good to have the I/O enabled if the chip and the data bus buffers aren't, we lenghtened the device select signal by delaying it and ANDing it with itself. We had no problems with this approach. (It is not a 'better' solution than Mr. Scouten's; he is quite right that one cannot use both the pin 41 signal and the $\overline{\phi 0}$ directly with the 6522 for their intended functions. The difference, however, between 180 and 50 ns required setup time makes it feasible to use the pin 41 decoded device select signal if one chooses.)

AN APPLE II PROGRAM EDIT AID

Alan G. Hill
12092 Deerhorn Dr.
Cincinnati, OH 45240

When editing an Apple Integer Basic program, you often want to locate all occurrences of a variable name, character string, or BASIC statements. This is usually the case when you are changing a variable name, moving a subroutine, etc., and you want to be sure you have located all references. The BASIC Edit program presented here should aid your editing.

The BASIC program should be loaded into high memory and the program to be edited appended to it. The Edit program uses a machine language routine at hex 300 to 39F to search BASIC statements for the requested string and return the BASIC line number in memory locations 17 and 18. The routine is re-entered at 846 to find the line number of the next occurrence. This process is continued until no further occurrences can be found. The high order byte of the line number (location 18) is set to hex FF to indicate that the search is finished.

BASIC Edit Program

Note in line 32680 of the BASIC program that LIST LINE is an invalid BASIC statement. You will have to resort to a little chicanery to get the statement in. First code line 32680 as PRINT LINE. Then, enter the monitor and change the PRINT token ($62) to a LIST token ($74). This is easiest done if you code line 32680 first and then search for the token in high memory ($3FFA when HIMEN is 16384).

After coding the BASIC program and the machine language routine, you will then need to append the program to be edited. Note that the program must have line numbers less than 32600. To append a program, you must first "hide" the Edit program. This is done by moving the HIMEN pointer (202) and (203) down below the Edit program. Then load the edited program and reset HIMEM: i.e.:

```
LOAD (EDIT PROGRAM)
POKE 76, PEEK (202)
POKE 77, PEEK (203)
LOAD (PROGRAM TO BE EDITED)
POKE 76,0 HIMEM MOD 256
POKE 77,64 HIMEM/256
```

You can then RUN 32600 the Edit program. Enter the character string or variable name to be searched when prompted by "FIND?". To search for a hex string (e.g. all occurrences of COLOR=), enter an @ character followed by the desired hex character pair (@66 for the COLOR= example)

EXAMPLES

To find all occurrences of:	**Input**
SCORE	SCORE
XYZ	XYZ
RETURN	@5B
DIM A	@4EC1
All references to 1000	@E803

The Edit program will end if the screen is full (> 18 lines). To continue the search for more occurrences, a RUN 32720 will return another page. Happy Editing!

Find Routine
Page Zero Memory Map

$3-4	Address of search limit. Set to HIMEM by routine, but could be set lower to avoid searching Edit program.
$6-7	Address of BASIC Token compared. Incremented until it exceeds Limit Address
$8-9	Ending address - 1 of current statement being scanned
$A-B	Address of string being searched. Set up by Edit program
$ C	Length - 1 of string being searched. Set up by Edit program
$11-12	Line number of statement containing the requested string. $12 is set to $FF if no more occurrences

```
FIND ROUTINE

A. G. HILL
MARCH 1979

HILO    *       $0003   HIMEM LO BYTE
HIHI    *       $0004   HIMEM HI BYTE
BSL     *       $0006   BASIC STATEMENT LO
BSH     *       $0007   BASIC STATEMENT HI
SEAL    *       $0008   STATEMENT ENDING ADDRESS LO
SEAH    *       $0009   STATEMENT ENDING ADDRESS HI
STRL    *       $000A   STRING LO
LNL     *       $0011   LINE NUMBER LO
LNH     *       $0012   LINE NUMBER HI
```

```
0300                    ORG   $0300

0300 A5 CA      START   LDA   $00CA   SET UP ADDRESS OF FIRST
0302 85 06              STA   BSL     BASIC STATEMENT IN
0304 A5 CB              LDA   $00CB   LOCS 6 AND 7
0306 85 07              STA   BSH
0308 A5 4C              LDA   $004C   SET UP TO STOP SEARCH
030A 85 03              STA   HILO    AT HIMEM.  COULD BE
030C A5 4D              LDA   $004D   CHANGED TO LIMIT SEARCH
030E 85 04              STA   HIHI    AT END OF PROGRAM BEING EDITED
0310 A0 00      LENGTH  LDYIM $00     GET STATEMENT LENGTH
0312 B1 06              LDAIY BSL
0314 38                 SEC
0315 E9 02              SBCIM $02     MINUS 2 TO POINT TO
0317 18                 CLC           LAST TOKEN IN STATEMENT
0318 65 06              ADC   BSL
031A 85 08              STA   SEAL    SET UP STATEMENT ENDING
031C A5 07              LDA   BSH     ADDRESS IN 8 AND 9
031E 69 00              ADCIM $00     ADD IN CARRY IF ANY
0320 85 09              STA   SEAH
0322 A0 01              LDYIM $01     SAVE LINE NUMBER IN
0324 B1 06              LDAIY BSL     IN 11 AND 12
0326 85 11              STA   LNL
0328 C8                 INY
0329 B1 06              LDAIY BSL
032B 85 12              STA   LNH
032D A2 00              LDXIM $00     ADJUST BSL TO POINT
032F A9 03              LDAIM $03     TO FIRST TOKEN
0331 20 64 03           JSR   INCPNT
0334 A0 00              LDYIM $00     COMPARE TOKEN TO
0336 B1 06      TTOKEN  LDAIY BSL     FIRST CHARACTER IN
0338 D1 0A              CMPIY STRL    STRING
033A D0 03              BNE   NXTOKN  IF NOT EQUAL POINT TO NEXT
033C 20 7F 03           JSR   COMPAR  IF EQUAL COMPARE REMAINING CHARS
033F 20 70 03   NXTOKN  JSR   INCTOK  POINT TO NEXT TOKEN
0342 90 F2              BCC   TTOKEN  CARRY CLEAR THEN LOOK AT NEXT
0344 A5 08              LDA   SEAL    AT END OF STATEMENT.
0346 C5 03              CMP   HILO    CHECK TO SEE IF AT END OF
0348 A5 09              LDA   SEAH    SEARCH LIMIT
034A E5 04              SBC   HIHI
034C B0 11              BCS   LIMIT   CARRY SET = LIMIT OF SEARCH
034E A5 08              LDA   SEAL    SET UP BSL AND BSH TO POINT
0350 85 06              STA   BSL     TO NEXT STATEMENT
0352 A5 09              LDA   SEAH
0354 85 07              STA   BSH
0356 A2 00              LDXIM $00     POINT TO LENGTH OF
0358 A9 02              LDAIM $02     STATEMENT BYTE
035A 20 64 03           JSR   INCPNT
035D D0 B1              BNE   LENGTH  ALWAYS BRANCH

035F A9 FF      LIMIT   LDAIM $FF     SET UP LARGE LINE NUMBER
0361 85 12              STA   LNH     TO INDICATE AT END OF SEARCH
0363 60                 RTS           RETURN TO BASIC
```

```
0364 18          INCPNT CLC             ROUTINE TO INCREMENT
0365 75 06              ADCX  BSL       POINTERS.  ENTER WITH
0367 95 06              STAX  BSL       XREG = DISPLACEMENT
0369 B5 07              LDAX  BSH       FROM
036B 69 00              ADCIM $00       BSL, BSH
036D 95 07              STAX  BSH       ACC = INCREMENT AMOUNT
036F 60                 RTS

0370 A5 06       INCTOK LDA   BSL       ROUTINE TO INCREMENT
0372 C5 08              CMP   SEAL      THE TOKEN ADDRESS BY 1
0374 A5 07              LDA   BSH       SET CARRY IF AT END
0376 E5 09              SBC   SEAH      OF STATEMENT
0378 E6 06              INC   BSL
037A D0 02              BNE   REXIT
037C E6 07              INC   BSH
037E 60          REXIT  RTS

037F A4 0C       COMPAR LDY   $000C     ROUTINE TO COMPARE
0381 B1 0A       COMPY  LDAIY STRL      REMAINING CHARACTERS
0383 D1 06              CMPIY BSL       (C) LENGTH OF CHARACTER
0385 F0 03              BEQ   COMPX     STRING -1
0387 A0 00              LDYIM $00       RESET YREG
0389 60                 RTS
038A 88          COMPX  DEY
038B 10 F4              BPL   COMPY     FOUND A MATCH! POP STACK ADDRESS
038D 68                 PLA             AND RETURN TO BASIC.  LINE NUMBER
038E 68                 PLA             IS ALREADY IN LNL AND LNH.
038F 60                 RTS
```

BASIC EDIT PROGRAM

```
32600 DIM A$(30)
32610 INPUT "FIND?",A$: CALL -936:
      IF A$(1,1)='@' THEN 32630:
      KK=LEN(A$): FOR I=1 TO KK:
      POKE 911+I,ASC(A$(I,I)): NEXT I
32620 POKE 12,KK-1: GOTO 32650
32630 A$=A$(2,LEN(A$)): KK=LEN(A$):
      FOR I=1 TO KK STEP 2:
      J=ASC(A$(I,I))-176:
      JJ=ASC(A$(I+1,I+1))-176
32640 IF J>9 THEN J=J-7:
      IF JJ>9 THEN JJ=JJ-7:
      POKE 912+I/2,J*16+JJ: NEXT I:
      POKE 12,KK/2-1
32650 POKE 10,912MOD256: POKE 11,912/256
32660 CALL 768
32670 IF PEEK(18)>127 THEN 32730:
      LINE=PEEK(17)+PEEK(18)*256
32680 LIST LINE
32690 IF PEEK(37)>18 THEN 32730
32700 CALL 846
32710 GOTO 32670
32720 CALL -936: GOTO 32700
32730 END
```

A CASSETTE OPERATING SYSTEM FOR THE APPLE II

Robert A. Stein, Jr.
2441 Rolling View Dr.
Dayton, OH 45431

Have you ever wished that as great as the Apple II computer system is that you were able load programs by name from a library cassette? Well, with this mini-sized cassette operating system you can stack many programs on one cassette and load the one you want by typing in its name. Great for showing off your system without juggling a dozen or so cassette tapes.

The **Cassette Operating System [CASSOS]** resides in memory at locations 02C0 to 03FF, where it won't get clobbered by BASIC programs or initalization. Add the optional cassette control circuit, or purchase one of the commercially available ones. (Candex Pacific, 693 Veterans BLVD, Redwood City, CA 94063) and you never need envy the PET for its loading technique again.

Operation

Load the 'CASSOS' tape, which you have created from the assembly listing, just like any other machine language program (2C0.3FFR), then initalize the BASIC pointers by depressing CTRL-B, return. To load a program depress CTRL-Y and RETURN. "PROG?" will be displayed, enter a 1-10 character program name. The cassette tape will be searched and the program loaded if found. "XXXXXXXXXX LOADED" will be output, where XXXXXXXXXX is the program now in memory. If the cassette control circuit (described later) is present the tape will also be stopped. A line of question marks (??????????) are displayed if the requested program was not found. To write a program to the library cassette enter Yc (Ctrl-Y, "WRITE", and RETURN. Program will be saved under the name requested at PROG? . "XXXXXXXXXX OUT" will be displayed at completion and the recorder stopped. To end a cassette program file enter: Yc, "EOF", RETURN; a special record header will be written. Note that to conserve limited memory space the EOF routine utilizes the program write subroutine so the "XXXXXXXXXX OUT" message should be ignored.

The program is structured such that the last 63 locations of the input buffer is used for display messages, so if more than 191 characters are entered at one time the program will still function, but without messages. The listing as presented was for a 16K system, change location 0358 as follows for a different configuration:

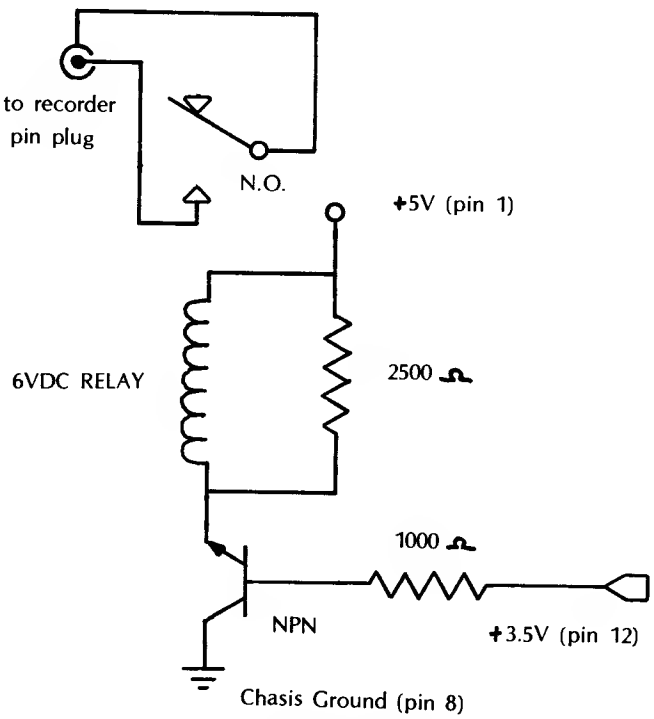
1F — 8K	5F — 24K
2F — 12K	7F — 32K
3F — 16K	8F — 36K
4F — 20K	BF — 48K

Program Design

The method by which CASSOS functions is to write a program header block consisting of header ID, program name, and start of the BASIC load. This is followed by the program data itself, utilizing the Apple monitor routines.

A Cassette On/Off Circuit

The following diagram describes a simple circuit for stopping and starting a cassette recorder which has a "remote" plug from the Apple II under program control. The theory involves activating or deactivating the AN3 signal on the Apple game connector. A store to location CO5F turns the recorder on and location CO5E turns it off. The strobe triggers a transistor which in turn opens a relay and closes the connection to the remote plug, starting the recorder. If your recorder requires an open connection to start tape movement wire the relay normally closed instead of open. It is also possible to add a relay that would interupt power to the recorder for control if you have no remote capability on your recorder.



Cassette Control Circuit

Parts List

All parts were purchased at a local Radio Shack
- 6VDC Relay (275-004)
- NPN Transistor (2N3568 or equivalent)
- 1000 Ohm Resistor
- 2500 Ohm Resistor
- Mini-Plug

All connections were made to a DIP Header which was modified by soldering a 16-pin IC to it so that the game paddles could still be used without modification when the cassette ON/Off circuit was in use. The common 6VDC relay was modified to be triggered by the game connector signals by wiring a 2500 ohm resistance (utilizing a series of resistors connected in series so that the sum is 2500 Ohms) in parallel with the relay coil. If your recorders rewind controls are disabled by the remote jack wire a switch to bypass the transistor between chasis ground and the relay, which will allow the rewind to operate when depressed. If all this is beyond your scope use the purchased control or simply stop and start the recorder manually.

```
02C0- A9 03     LDA  #$03
02C2- 8D A0 02  STA  $02A0
02C5- A9 A1     LDA  #$B1
02C7- 20 67 03  JSR  $0367
02CA- A9 FF     LDA  #$FF
02CC- 8D BB 02  STA  $02BB
02CF- A5 CA     LDA  $CA
02D1- 8D BC 02  STA  $02BC
02D4- A5 CB     LDA  $CB
02D6- 8D BD 02  STA  $02BD
02D9- 20 CD FE  JSR  $FECD
02DC- A4 CA     LDY  $CA
02DE- A5 CB     LDA  $CB
02E0- 20 60 03  JSR  $0360
02E3- 20 CD FE  JSR  $FECD
02E6- A9 EB     LDA  #$EB
02E8- 20 7E 03  JSR  $037E

02EB- 87 A0 CF D5 99
02F0- FF 87 A0 CC CF C1 C4 C5
02F8- C4 FF 10 D2 CF C7 BF FF


0300- A2 A2     LDX  #$02
0302- D0 07     BNE  $030B
0304- 84 60     STY  $60
0306- 20 62 FC  JSR  $FC62
0309- A4 60     LDY  $60
030B- 8E 15 03  STX  $0315
030E- 8C 14 03  STY  $0314
0311- A0 00     LDY  #$00
0313- B9 FA 02  LDA  $02FA,Y
0316- C9 FF     CMP  #$FF
0318- F0 2D     BEQ  $0347
031A- 20 ED FD  JSR  $FDED
031D- C8        INY
031E- 10 F3     BNE  $0313
0320- 48        PHA
0321- A9 A2     LDA  #$02
0323- 86 60     STX  $60
0325- 8C 61     STY  $61
0327- A9 A0     LDA  #$A0
0329- 20 6C FD  JSR  $FD6C
032C- 68        PLA
032D- AA        TAX
032E- A0 00     LDY  #$00
0330- B9 A0 02  LDA  $02A0,Y
0333- C9 8D     CMP  #$8D
0335- F0 08     BEQ  $033F
0337- 91 60     STA  ($60),Y
0339- C8        INY
033A- CA        DEX
033B- F0 AA     BEQ  $0347
033D- 10 F1     BNE  $0330
033F- A9 A0     LDA  #$A0
0341- 91 60     STA  ($60),Y
0343- C8        INY
0344- CA        DEX
0345- D0 F8     BNE  $033F
0347- 60        RTS
0348- A0 10     LDY  #$10
034A- A2 00     LDX  #$00
034C- 20 51 03  JSR  $0351
034F- A0 B1     LDY  #$B1
0351- A9 02     LDA  #$02
0353- D0 04     BNE  $0359
0355- A0 FF     LDY  #$FF
0357- A9 3F     LDA  #$3F
0359- 95 3D     STA  $3D,X
035B- 94 3C     STY  $3C,X
035D- E8        INX
035E- E8        INX
035F- 60        RTS
0360- A2 00     LDX  #$00
0362- 20 59 03  JSR  $0359
0365- D0 EE     BNE  $0355
0367- 85 50     STA  $50
0369- A2 02     LDX  #$02
036B- A0 FA     LDY  #$FA
036D- 20 04 03  JSR  $0304
0370- 20 48 03  JSR  $0348
0373- A9 0A     LDA  #$0A
0375- A6 50     LDX  $50
0377- 20 20 03  JSR  $0320
037A- 8D 5F C0  STA  $C05F
037D- 60        RTS
037E- 48        PHA
037F- 8D 5E C0  STA  $C05E
0382- A2 02     LDX  #$02
0384- A0 B1     LDY  #$B1
0386- 20 04 03  JSR  $0304
0389- 68        PLA
038A- A8        TAY
038B- 20 00 03  JSR  $0300
038E- 4C 03 E0  JMP  $E003
0391- A9 A3     LDA  #$A3
0393- 20 67 03  JSR  $0367
0396- 20 48 03  JSR  $0348
0399- 20 FD FE  JSR  $FEFD
039C- AD B0 02  LDA  $02B0
039F- C9 D3     CMP  #$D3
03A1- D0 29     BNE  $03CC
03A3- AC BC 02  LDY  $02BC
03A6- AD BD 02  LDA  $02BD
03A9- 20 60 03  JSR  $0360
03AC- 20 FD FE  JSR  $FEFD
03AF- A2 00     LDX  #$00
03B1- BD B1 02  LDA  $02B1,X
03B4- DD A3 A2  CMP  $02A3,X
03B7- D0 DD     BNE  $0396
03B9- E8        INX
03BA- E0 0A     CPX  #$0A
03BC- D0 F3     BNE  $03B1
03BE- AD BC 02  LDA  $02BC
03C1- A5 CA     STA  $CA
```

```
03C3-   AD BD 02    LDA   $02BD
03C6-   85 CB       STA   $CB
03C8-   A9 F1       LDA   #$F1
03CA-   D0 B2       BNE   $037E
03CC-   8D 5E C0    STA   $C05E
03CF-   A2 20       LDX   #$20
03D1-   A9 BF       LDA   #$BF
03D3-   20 ED FD    JSR   $FDED
03D6-   CA          DEX
03D7-   D0 F8       BNE   $03D1
03D9-   20 DD FB    JSR   $FBDD
03DC-   F0 B3       BEQ   $0391
03DE-   AD 01 02    LDA   $0201
03E1-   C9 D7       CMP   #$D7
03E3-   F0 10       BEQ   $03F5
03E5-   C9 C5       CMP   #$C5
03E7-   D0 A8       BNE   $0391
03E9-   8D B0 02    STA   $02B0
03EC-   20 48 03    JSR   $0348
03EF-   8D 5F C0    STA   $C05F
03F2-   4C CA A2    JMP   $A2CA
03F5-   4C C0 02    JMP   $02C0
03F8-   4C DE 03    JMP   $03DE
03FB-   00          BRK
03FC-   00          BRK
03FD-   00          BRK
03FE-   00          BRK
03FF-   00          BRK
```

A Cassette Tape Catalog

Shown in exhibit is a short integer BASIC program which when loaded will list all the programs on a CASSOS format library tape. The CASSOS sub-routines are used so the software must be core resident. Just load the program, insert the library cassette into the cassette handler, and type RUN after starting the cassette player.

```
 10 N=1: CALL -936: VTAB (10): DIM X$(1)
 20 INPUT "INSERT LIBRARY TAPE AND DEPRESS 'RETURN'";X$
 30 POKE -16289,0: CALL -936: GOSUB 300
 40 PRINT "FILE #  PROGRAM NAME  BYTES"
 50 PRINT "------  ------------  -----"
 60 CALL 840: CALL -259
 70 IF PEEK (688)= ASC("E") THEN 210
 80 IF PEEK (688)# ASC("S") THEN 200
100 REM  LOAD INTO NON-EXIST MEMORY (800-BFF)
110 POKE 60, PEEK (700): POKE 61,( PEEK (701)+128)
120 POKE 62,255: POKE 63,191: CALL -259
130 PRINT N;: POKE 789,2: POKE 788,177: CALL 785
140 L= PEEK (700)+ PEEK (701)*256
150 L=16384-L:N=N+1
160 PRINT "     ";L: GOTO 60
200 GOSUB 300: PRINT "NO EOF MARK"
210 POKE -16290,0: GOSUB 300
230 PRINT "***END OF FILE***"
240 CALL -155
300 FOR I=1 TO 30
305 L= PEEK (-16336)+ PEEK (-16336): NEXT I
310 CALL -1059: RETURN
```

```
]RUN
INSERT LIBRARY TAPE AND DEPRESS 'RETURN'

FILE #  PROGRAM NAME  BYTES
------  ------------  -----
1DIRECTORY      544
2BILLBOARD      238
3R.ROULETTE     530
4COLORBYROD     185
5HELLO          2830
6BOWLING        2119
7BOXING         2636
8TICTACTOE      3461
***END OF FILE***
```

S-C ASSEMBLER II
Super Apple II Assembler

Chuck Carpenter
2228 Montclair Pl.
Carrollton, TX 75006

I've had the good fortune to get an advance copy of an excellent assembler for the Apple II. The assembler was written by Bob Sander-Cederlof and has many desireable features. Bob has used sweet 16 and several routines from the monitor and integar BASIC (it doesn't run with the Applesoft ROM on). The result is a compact co-resident two-pass assembler. A summary of assembler commands and data is listed in Table 1.

Here are a few of the assembler features:

- Format compatible with Apple mini-assembler
- Complete text editing using standard Apple screen and line editing features.
- Save and Load as in integar BASIC
- Psuedo op codes
- Text for REMs following the line no.
- Tabs to the opcode, operand and comment field using (CTRL) I
- Symbol table
- Listing, fast or slow
- Stop and start a LIST or ASM at any time
- Access Apple monitor from the assembler using $
- Run programs from the assembler

The S-C ASSEMBLER II includes many other features. Among these are:

- Line renumbering starting at 1000 by 10's
- Printer driver routine - his or yours (or mine for that matter).
- Pagination of printed output
- Program location and relocation
- Can be used to renumber BASIC programs (except branches)
- Operates within DOS (see Table 2)
- Runs on an 8K machine

I have included a couple of examples of the S-C ASSEMBLER II features in Figure 1 and 2. Figure 1 is a functional routine. Figure 2 is merely for illustration of the .DA feature. Most of the assembler capability is illustrated in Figure 1. This routine, which compares 2 byte data, can be used for many applications such as extended loop counters. The example also includes ASCII strings using the pseudo op code .AS.

A jump to the user exit at $3F8 was used to enter the data. This also takes advantage of the (CTRL) Y feature of the Apple monitor.

By calling the print routine with PRT, a hard copy of a listing or of assembled output is obtained. The printer driver routine is output from the game paddle connector. This is a TTL level serial signal. Typing SLO(W) or FAS(T) stops the printer output. Also, SLO(W) will provide a slow listing of your program. You can stop and start the listing with the space bar and, escape back to the assembler with a (RETURN). FAS(T) cancels SLO(W) returning to normal screen speed. (See Slow List, MICRO #5 page 21.)

For text editing, you can insert a line between other lines and list any single line or combination of lines. This allows character editing or line editing using Apple ESCAPE functions ((ESCAPE)D,C,B). Also you can DEL(ETE) any line or combination of lines.

An asterisk (*) in the first column of the label field allows that line to be a comment or blank line. Very useful for commenting a program. I used short comments in my programs; I only have 48 columns. Actually the comment can be any length (up to 100 characters or so). An asterisk used in the operand field means current location. You can add or subtract labels, hex and decimal values from the current location. Each of these can be added or subtracted, to or from, each other. Here are some examples:

```
1000 LABL LDA *-*  CURRENT-CURRENT
1010 LAB2 LDA LABL-LABL
1020 LAB3 LDA *-LABL
1030 LAB4 LDA LABL+1234
1040 LAB5 LDA $1234-LABL
1050 LAB6 LDA $ABCD-5678
1060 *
1070 * EXAMPLES OF ADDITION & SUBTRACTION OF
1080 * CURRENT VALUE, LABELS, DECIMAL AND
1090 * HEX VALUES FROM EACH OTHER.
1100 *
```

Illustration of the .DA feature is shown in Figure 2. The intent here is to show data in a single or 2 byte location. Once the data value has been assigned with the .DA code, it can be manipulated with another feature. This feature is shown as a / (slant line) and # (pound) in the first column of the operand field. Here's what's happening:

```
LDA /LABL = HIBYTE = ÷256
LDA #LABL = LOBYTE = MOD256
```

As you can see from this and the previous examples, these features provide a very powerful assembler capability.

Before I obtained this assembler I could never get very enthusiastic about extensive machine or assembly language programming. Now, with this assembler, this coding is as easy as BASIC. You can get a copy for your Apple II from:

S-C SOFTWARE
P.O. Box 5537
Richardson, TX 75080
Price - $25.00

I think you will enjoy it: having the efficiency of machine language programs developed with the ease of BASIC. The combination of compact programs with interactive capability makes personal computing even more enjoyable.

Load: *1000.1CFFR
Run: *1000G Hard Entry
or: *1003G Soft Entry

Pseudo ops:

label .OR expr	origin (optional label)
label .EQ expr	equate
label .DA expr	data (optional label)
label .HS xxxx...x	hex string
label .AS daaaa...ad	ascii string (d is any delimiter)
.EN	end

Commands:

LOAD	load program from tape
SAVE	save program to tape
LIST	list entire program
LIST line#	list selected line
LIST line#,line#	list range of lines
DELETE line#	delete selected line
DELETE line#,line#	delete range of lines
RENUMBER	renumbers all lines
NEW	erase program
SLOW	program slow list
FAST	program fast list
PRT	printer driver $1B77-1BFF
ASM	assemble program
RUN expr	execute starting at expr
APPEND	add program from tape to one in memory

Table 1
S-C Assembler II Summary Notes

Instruction Steps:

1. Bring up DOS per instruction manual
2. Reset to monitor (*)
3. Load assembler from tape
4. Return to DOS using $3DOG
5. BSAVE Assembler
6. LOCK Assembler
7. Call 4096 Jumps to Assembler
8. $3DOG Jumps to DOS soft entry but...

At this point the DOS is clobbered. Any further use of DOS requires a reboot. It is very handy though to have the speed of loading the assembler from the disc.

Table 2
S-C Assembler II with Apple II DOS

```
:ASM                    1000 * .DA PSEUDO OP EXAMPLE
                        1010 *
                        1020       .OR $300
0300- 34 12             1030 HEX   .DA $1234
0302- 34 12             1040 DEC   .DA 4660
                        1050 *
                        1060 * ADDRESS OF DATA
                        1070 *
0304- A9 00             1080       LDA #HEX      HEX LO BYTE
0306- A9 03             1090       LDA /HEX      HEX HI BYTE
                        1100 *
                        1110 * DATA AT THE ADDRESS
                        1120 *
0308- AD 02 03          1130       LDA DEC       DEC LO BYTE
030B- AD 03 03          1140       LDA DEC+1     DEC HI BYTE

                        1150       .EN

SYMBOL TABLE

HEX   0300    DEC   0302

:
```

Figure 2
DA Pseudo Op Example

```
:NEW
S-C ASSEMBLER ][


:LOAD

:ASM
```

```
1000 * S-C ASSEMBLER II EXAMPLE
1010 *
1020 * COMPARES HEX VALUES
1030 * AND INDICATES WHICH
1040 * IS GREATER (OR EQUAL).
1050 *
1060       .OR $300
1070 *
1080 *  .OR DEFAULT IS $0800
1090 *
1100 COUT .EQ $FDED
0300- 58 20 3C
0303- 20 59     1110 LESS .AS 'X < Y'
0305- 8D        1120      .HS 8D
0306- 58 20 3E
0309- 3D 20 59  1130 GREQ .AS 'X >= Y'
030C- 8D        1140      .HS 8D
                1150 XL   .EQ $3C
                1160 XH   .EQ $3D
                1170 YL   .EQ $3E
                1180 YH   .EQ $3F
030D- A5 3C     1190 STAR LDA XL
030F- C5 3E     1200      CMP YL
0311- A5 3D     1210      LDA XH
0313- E5 3F     1220      SBC YH
0315- B0 06     1230      BCS TST1      X >= Y
0317- A0 00     1240      LDY #LESS-LESS
0319- 20 28 03  1250      JSR PRNT
031C- 60        1260      RTS
031D- A0 06     1270 TST1 LDY #GREQ-LESS
031F- 4C 28 03  1280      JMP PRNT
0322- 09 80     1290 PRT1 ORA #$80      NORMAL OUT
0324- 20 ED FD  1300      JSR COUT
0327- C8        1310      INY
0328- B9 00 03  1320 PRNT LDA LESS,Y
032B- 10 F5     1330      BPL PRT1
032D- 4C ED FD  1340      JMP COUT
                1350 *
                1360 * DATA ENTRY THROUGH
                1370 * USER EXIT @ $3F8.
                1380 *
                1390 * DATA.DATA (CTRL)Y
                1400 *
                1410      .OR $3F8
03F8- 4C 0D 03  1420      JMP STAR
                1430      .EN
```

```
:
:EXAMPLE RUN
:
:$10000.20000
X >= Y

:$2000.1000
X >= Y

:$3.3
X >= Y

:
```

SYMBOL TABLE

COUT	FDED	LESS	0300	GREQ	0306
XL	003C	XH	003D	YL	003E
YH	003F	STAR	030D	TST1	031D
PRT1	0322	PRNT	0328		

Figure 1
S-C Assembler II Example

THE INTEGER BASIC TOKEN SYSTEM IN THE APPLE II

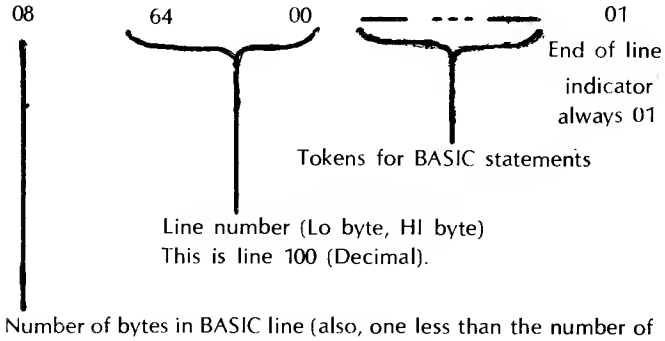
Frank D. Kirschner
2643 Rockledge Trail
Dayton, OH 45430

There are two primary methods of storing BASIC programs in microcomputers. One involves storing the entire program, letter by letter and symbol by symbol somewhere in memory, and interpreting the ASCII codes on execution. This is typical of BASIC compilers and some interpreters, like the TRS-80 Level 1. A more memory-efficient system uses tokens, eight bit bytes each of which represent a BASIC word or symbol. The TRS-80 Level II uses this method, as does the Apple II, to which the examples which follow apply.

When in Integer BASIC, the Apple stores characters as they are entered in a character buffer (hex locations 0200 to 02FF). When "return" is entered, BASIC "parses" the entry (that is, interprets the ASCII characters and breaks the instruction into executable parts). It determines what is a command, what are variables, data and so forth. If it is legal and is preceded by a number between 0 and 32767 (a line number), it stores it in memory in a fashion discussed below. If there is no line number, it simply executes the command and awaits further instructions.

The way the programs are stored is quite clever. When BASIC is initiated (control B or E000 G from the monitor) several things happen. First, the highest available user memory (RAM) is stored in memory locations 004C (Lo byte) and 004D (Hi byte), called the HIMEM pointer. Also, locations 00CA and 00CB, the start-of-program pointer, get the same numbers, since there is no program as yet. As program steps are entered, they are stored starting at the top of memory, highest line numbers first, and the start-of-program pointer is decreased accordingly. See Figure 1. When a line with a higher number than some already in memory is entered, they are shuffled to preserve the order. One application: if you enter a program and then hit control B, the program is **not** scratched (or erased); only the start-of-program pointer is affected. Since powering up the Apple fills the memory with a pattern of ones and zeros (it looks like FF FF 00 00 ...) from the monitor, it is easy to find the start of the program and then manually reset CA and CB to that location.

This is the way program instructions are stored in memory: (All numbers are in hex)



Number of bytes in BASIC line (also, one less than the number of bytes from the beginning of the next line.

Figure 2

As an example, power up the Apple, bring up BASIC, and enter

100 PRINT 0,50

Enter the monitor (by pushing "reset"), and then examine the program by entering

EXAMPLES FOR
16K Apple

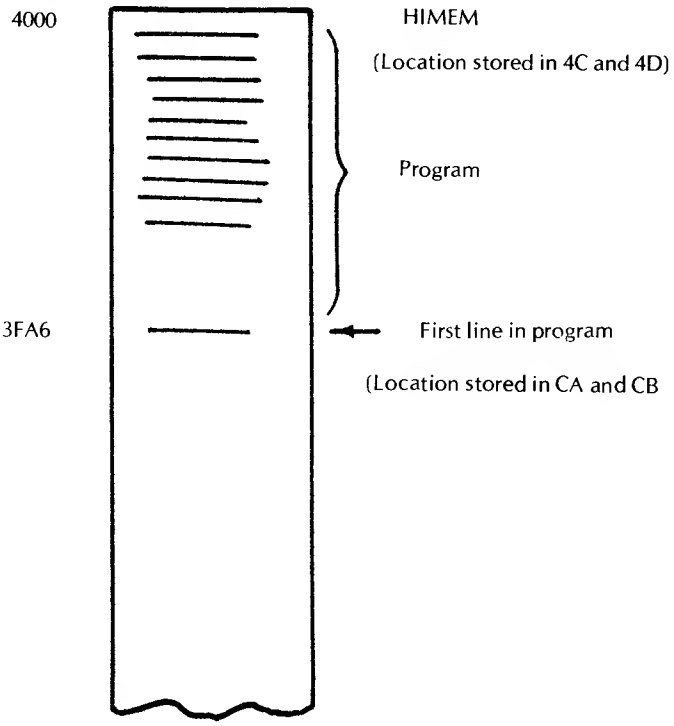


Figure 1

Memory Map for Program Storage

3FF4.3FFF return

(Locations for a 16K Apple. Subtract 2000 hex for a 4K or add 4000 hex for a 32K Apple.) You will see this:

3FF4 - 0C 64 00 62
3FF8 - B0 00 00 49 B5 32 00 01

which means:

0C	There are 12 bytes in this line
64 00	It is line 100 (Decimal)
62	PRINT (see Table 1 for complete list of tokens
B0	The next two bytes are a number (rather than tokens)
00 00	The number 0
49	The comma in a PRINT statement
B5	Another number follows
32 00	The number 50
01	End of BASIC line

To demonstrate the use of this information, return to BASIC and try to enter the following BASIC line:

100 DEL 0,50

You will get a syntax error, because the Apple Interpreter does not allow the command DEL in deferred execution mode. Now do this: reenter the monitor and change the 62 (PRINT) to 09 (DEL) and the 49 (,for PRINT) to 0A (, for DEL) by entering

3FF7: 09 Return

3FFB: 0A Return

Reenter BASIC (control C) and list. Try this instruction by adding lines between 0 and 50, running the program, and then listing it. This allows you to write a program which will carry out some functions only the first time it is run and then automatically delete those lines.

In addition to inserting instructions which cannot be entered as deferred commands, you can modify the program under program control. As an example, here is a program which will stop and start listing a long program by hitting a key on the keyboard.

Bring up BASIC.

Enter: 257 LIST 0: RETURN

HIT RESET, 3FF6.3FFF RETURN

You will see

3FF6 - 0A 01

3FF8 - 01 74 B0 00 00 03 5B 01

What this means:

3FF6: 0A Ten bytes in line

3FF7,8: 01 01 LINE 257

3FF9: 74 TOKEN FOR LIST

3FFA: B0 Means "Number follows"

3FFB,C: 00 00 LINE TO BE "LISTED" (LO, HI)

3FFD: 03 TOKEN FOR COLON

3FFF: 01 End of BASIC LINE

Now enter 3FF7: FF FF Return

Cont. C, List

You have 65535 LIST O: RETURN

Now enter

100 X=PEEK (-16384): POKE -16368, 0:1F

X 127 THEN 0: GOTO 100

Reset, 3FCF.3FFF Return

Change line no. from 100 to 65534 by entering 3FDO; FE FF Return

Change GOTO 100 to GOTO 65534 by entering 3FF3: FE FF

Change the 0 in "THEN 0" to 65533 by entering 3FEE: FD FF

In like manner, enter these remaining steps: (Under each number which has to be entered through the monitor, the Hex equivalent, in reverse order as it must be entered, appears)

65533 I = I PEEK (I): IFI> PEEK (76)+
(FD FF)
256*PEEK (77) THEN END: GOTO
65531
(FB FF)

65532 X=PEEK (-16384):POKE -16386,0:
(FC FF)
IF X 127 THEN 65534
(FE FF)

65531 POKE 16374, PEEK (I †1): POKE 16380
(FB BB)
PEEK (I+2): GOSUB 65535
(FF FF)

'32767 I=PEEK (202) 256* PEEK (203)

The steps must be entered in reverse order (i.e descending line numbers) because the interpreter orders them by their number when entered, and will not re-order lines when the numbers have been changed through the monitor.

The reason for making all these line numbers very high is so the applications program will fit "under" the list program.

Now, in the monitor, move the start of program and HIMEM pointers below the program:

3A: 49 3F Return

4C: 49 3F Return

Hit control C and list. Nothing is listed. The program has been stored in a portion of memory temporarily inaccessible to BASIC. Load your applications program, make sure all the line numbers are less that 32767, and change HIMEM through the monitor (4C: 00 40) and execute RUN 32767. The program will list until you hit a key and then resume when you hit a key again. It uses the fact that each line begins with the number of bytes in the line followed by the line number. Numbers of successive lines are found and "POKE into the appropriate location in line 75535, which then lists each line.

Using these methods you can exercise considerably more control over the BASIC interpreter in your microcomputer.

IMPROVED STAR BATTLE SOUND EFFECTS

William M. Shryock, Jr.
P.O. Box 126
Williston, ND 58801

```
10  POKE 0,160: POKE 1,1: POKE
    2,162: POKE 3,0: POKE 4,138
    : POKE 5,24: POKE 6,233: POKE
    7,1: POKE 8,208: POKE 9,252
    : POKE 10,141
20  POKE 11,48: POKE 12,192: POKE
    13,232: POKE 14,224: POKE 15
    ,150: POKE 16,208: POKE 17,
    242: POKE 18,136: POKE 19,208
    : POKE 20,237: POKE 21,96
30  CALL -936: VTAB 12: TAB 9: PRINT
    "STAR BATTLE SOUND EFFECTS"

40  SHOTS= RND (15)+1
50  LENGTH= RND (11)*10+120
60  POKE 1,SHOTS: POKE 15,LENGTH:
    CALL 0
70  FOR DELAY=1 TO RND (1000): NEXT
    DELAY
80  GOTO 40
```

This version can be used in low res. programs without having to reset HIMEM. Also it can be loaded from BASIC.

TABLE I

APPLE II INTEGER BASIC TOKENS

BASIC COMMAND OR FUNCTION	HEX TOKEN	
ABS	31	
(	3F	
)	72	
ASC (	3C	Includes left paren.
)	72	
"	28	first quote
"	29	second quote
AUTO	0D	
,	0A	
CALL	4D	
CLR	0C	
COLOR=	66	Includes =
CON	60	
DEL	09	
,	0A	
DIM	4F	Numeric Arrays
(	34	
)	72	
DIM	4E	String Array
(	22	
)	72	
$	40	
DSP	7C	Numeric Variable
DSP	7B	String Variable
END	51	
FOR	55	
=	56	
TO	57	
STEP	58	
GOSUB	5C	
GOTO	5F	
GR	4C	
HIMEN:	10	Includes :
HLIN	69	
,	6A	
AT	6B	
IF	60	
THEN	24	When followed by a line no.
THEN	25	When followed by GOSUB or a basic operation
INPUT	54	Numberic Variable
INPUT	52	String Variable
INPUT	53	Input if followed by ...
,	27	
"	28	first
"	29	Second
IN #	7F	Includes #
LEN (	3B	Includes (
LET	5E	
LIST	74	
,	75	

BASIC COMMAND (CONT)	HEX TOKEN	
LOAD	04	
MAN	0F	
NEW	0B	
NEXT	59	
,	5A	
NO DSP	79	
NO TRACE	7A	
PDL	32	
(	3F	
)	72	
PEEK	2E	
3F	(	
72	)	
PLOT	67	
,	68	
POKE	64	
,	65	
POP	77	
PRINT	63	If used alone
PRINT	62	Numeric Variable
:	46	
,	49	
PRINT	61	String Variable
"	28	First
"	29	Second
PR #	7E	Includes #
REM	5D	
RETURN	5B	
RND	2F	
(	3F	
)	72	
-	36	
SAVE	05	
SCRN (	3D	Includes (
,	3E	
)	72	
SGN	30	
(	3F	
)	72	
TAB	50	
TEXT	4B	
TRACE	7D	
VLIN	6C	
,	6D	
AT	6E	
VTAB	6F	
:	03	
=	71	In assignment
AND	1D	
OR	1E	
MOD	1F	
NOR	DE	

Chuck Carpenter
2228 Montclair Place
Carrollton, TX 75006

Renumbering Applesoft programs suddenly became possible. The resequence program in Jim Butterfield's "Inside Pet BASIC," (MICRO 8:39) solved the problem.

After clearing up a minor problem in the program (with help from Jim) I tried it on a 200 line program. Because of the way I started numbering in the first place, I had to fix-up about a dozen lines. But, I never would have gotten through that much renumbering otherwise.

As Jim mentioned in his letter to me, a machine language program would have ran a whole bunch faster. With DOS and having to find a place to locate such a program, the BASIC approach may be easier.

Here are some comments on the Applesoft version shown in Listing 1:

-Line 60005 has some prompting inputs to set-up the program.
-Use RUN 60005 to start renumbering.
-Line 60060 brances to a DELete line.
-Line 60160 is changed to a point to the line no. in Applesoft (2049 or $801).

Note: These are the pointers for Applesoft ROM

-Line 60160 was also changed to allow starting at any line number (M=LN-IN).
-Line 60170 changed to allow any numbering increment (M=M †IN).
-Line 60220 - tokens changed for Applesoft (this information is in the Applesoft II manual).
-Line 60260 and 60270 added to delete the renumber program and end it.

To make using the program easier, an append program (also for ROM) does the job. The assembly language program shown in listing 2 links the two programs together. You only need to do this if you want to renumber an existing program. (You can still load the renumber program before you start a new program.) Here's how you use it.

-Load the append program first. It fits in page 3 starting at $3A5.
-Load the lower line no. Applesoft program.
-Type Call 933 and (return).
-Load the higher line no. renumber program.
-Type CALL 955 and (return).
-Use RUN 60005 to start renumbering.

Be sure to record any output that appears on the screen. Write down the information and check the renumbering on the lines indicated. Putting longer line numbers in short spaces will be one message. Another will ask you to check where you used a THEN for a GOTO. The renumber program is not sure if it should renumber a line or a parameter.

My thanks to Jim Butterfield for providing us with such a useful program (and helping me get this one running). Also, thanks to Bob Matzinger from the Dallas Area Apple Corps for some modification suggestions and the Applesoft ROM append routine.

```
*
*03A5L

03A5-   A5 67     LDA   $67
03A7-   85 06     STA   $06
03A9-   A5 68     LDA   $68
03AB-   85 07     STA   $07
03AD-   38        SEC
03AE-   A5 69     LDA   $69
03B0-   E9 03     SBC   #$03
03B2-   85 67     STA   $67
03B4-   A5 6A     LDA   $6A
03B6-   E9 00     SBC   #$00
03B8-   85 68     STA   $68
03BA-   60        RTS
03BB-   A5 06     LDA   $06
03BD-   85 67     STA   $67
03BF-   A5 07     LDA   $07
03C1-   85 68     STA   $68
03C3-   20 F2 D4  JSR   $D4F2
03C6-   60        RTS
03C7-   FF        ???
03C8-   FF        ???
*
```

Listing 2

Applesoft append program. This program can be used to append any two programs together.

```
]LIST

60000  END
60005  HOME : PRINT : PRINT "RENUMBER:": PRINT : I
NPUT "FIRST LINE # - ";LN: PRINT : INPUT "INCREMEN
T - ";IN
60010  LET T = 0: DIM V%(100),W%(100): GOSUB 60160
: FOR R = 1 TO 1E3: GOSUB 60210
60020  IF G THEN  GOSUB 60090: NEXT R
60030  GOSUB 60160: FOR R = 1 TO 1E3:N =  INT (M /
 256): POKE A - 1,M - N * 256
60040  POKE A,N:V = L: GOSUB 60070:W%(J) = M: GOSU
B 60170: IF G THEN  NEXT R
60050  GOSUB 60160: FOR R = 1 TO 1E3: GOSUB 60210:
 IF G THEN  GOSUB 60110: NEXT R
60060  PRINT "*END*": GOTO 60260
60070  LET J = 0: IF T <  > 0 THEN  FOR J = 1 TO T
: IF V%(J) <  > V THEN  NEXT J:J = 0
60080  RETURN
60090  IF V <  > 0 THEN  GOSUB 60070: IF J = 0 THE
N T = T + 1:V%(T) = V
60100  RETURN
60110  GOSUB 60070: IF J = 0 THEN  RETURN
60120 W = W%(J): IF W = 0 THEN  PRINT "GO";"L";L;"
?": RETURN
60130  FOR D = A TO B + 1 STEP  - 1:X =  INT (W /
10):Y = W - 10 * X + 48: IF W = 0 THEN Y = 32
60140  POKE D,Y:W = X: NEXT D: IF W = 0 THEN  RETU
RN
60150  PRINT "INSERT";W%(J);"L";L: RETURN
60160  LET F = 2049:M = LN - IN
60170  LET A = F:M = M + IN
60180  LET F =  PEEK (A) +  PEEK (A + 1) * 256:L =
  PEEK (A + 2) +  PEEK (A + 3) * 256:A = A + 3:G =
 L < 6E4
60190  RETURN
60200  LET S = 0
60210  LET V = 0:A = A + 1:B = A:C =  PEEK (A): IF
 C = 0 THEN  GOSUB 60170: ON G + 2 GOTO 60210,6019
0
60220  IF C <  > 171 AND C <  > 176 AND C <  > 196
 AND C <  > S GOTO 60200
60230  LET A = A + 1:C =  PEEK (A) - 48: IF C =  -
 16 GOTO 60230
60240  IF C >  = 0 AND C < 9 THEN V = V * 10 + C:
GOTO 60230
60250  LET S = 44:A = A - 1: RETURN
60260  DEL 60000,60270
60270  END

]
```

Listing 1

APPLE II Applesoft Version of Jim Butterfield's Resequence program.

AN APPLE II PROGRAM RELOCATOR

Rick Auricchio
59 Plymouth Avenue
Maplewood, NJ 07040

After writing an Assembly-language program, the occasion often arises when one wishes the program to run in a different area of memory than that for which it was originally assembled. Relocating a program requires changing all absolute references within the program, so that it will run elsewhere in memory...this process is tedious, time-consuming, and repetitive WORK.

ENTER THE ELECTRONIC BRAIN

Behold! We have before us an electronic marval which thrives on such repetitive work! After all, why not just write a program to relocate others? Read on.....

HERE'S WHAT IT TAKES

When a Relocating Assembler creates object code one of the items built is a Relocation Dictionary. This is actually a table of pointers to the program instructions that have absolute addresses; it also contains some flags for use by a relocating loader so that the latter can adjust the address references during the loading process.

Unfortunately, we don't have such a luxury when relocating most programs...all we have is raw machine language to work with. Our relocator will have to scan the subject program and find all absolute references which need adjustment.

FUNCTIONAL DESCRIPTION of RELOC8

The RELOC8 program will use the Apple's SWEET-16 utility for all 16-bit data and address manipulation; use of SWEET-16 saves a lot of 6502 code at the expense of some speed loss. In order to decipher the 6502 instructions of the subject program, Apple's Disassembler is used. (The disassembler, by the way, turns out to be a rather nice utility for things like this). In order to minimize user intervention, it was decided that RELOC8 would be run as part of a standard Apple Memory-Move command. After loading the subject program in its "old" memory location, one enters an Apple Move command to copy it to the "new" memory location, followed by Control-Y (which starts RELOC8 after the Move completes).

All absolute address references which lie within the range of the subject program will be updated. References to addresses outside the subject program (e.g. for Monitor calls) need not be changed.

USING RELOC8

To relocate a machine-language program, the following procedure is followed: load RELOC8 into the Apple and load the subject program into its "old" location. Type an Apple Move command to move the subject program to its "new" address followed by a space and control-Y. The RELOC8 program will print all modified instructions and then exit when it's done. For example, to relocate a subject program from "old" location 1500-1800, to "new" location 2A00-2D00, one would type the following command:

```
* 2A00<1500.1800M Yc
```

This is a standard "move" command, moving the program with the Apple Monitor; however, we follow the "M" with a space and a control-Y so that RELOC8 will be entered immediately following the move command. When it is entered, RELOC8 picks up the address values from the "move" command.

A FEW WORDS OF WARNING

There is something to watch out for while using RELOC8. Since it scans the subject program for absolute addresses, any data imbedded within the program may cause RELOC8 to think the data is an instruction. In that case, the data will be modified and RELOC8's opcode scan might get "out of sync" with the real instructions in the subject program. It's best to try and keep data separate from instructions; if RELOC8 does modify some data, you'll have to fix it before running the relocated program.

```
*****************************
*                           *
*  MACHINE-LANGUAGE         *
*   PROGRAM RELOCATOR       *
*                           *
*     -- RELOC8 --          *
*                           *
*  RICK AURICCHIO 10/26/78  *
*                           *
*   FOR THE APPLE-II        *
*                           *
*****************************
*
* --- SWEET-16 REGISTERS
*
AC       EQU      0    R0:ACCUMULATOR
OB       EQU      1    R1:OLD BASE
OE       EQU      2    R2:OLD END
NB       EQU      3    R3:NEW BASE
NE       EQU      4    R4:NEW END
RB       EQU      5    R5:RELOCATION BIAS
*
```

```
00000000            ACL     EQU     0
00000001            ACH     EQU     1
00000002            OBL     EQU     2
00000003            OBH     EQU     3
00000004            OEL     EQU     4
00000005            OEH     EQU     5
00000006            NBL     EQU     6
00000007            NBH     EQU     7
00000008            NEL     EQU     8
00000009            NEH     EQU     9
                    *
                    *
0000F689            SWEET16 EQU     X'F689'     SWEET-16 INTERPRETER
0000F88E            INSDS2  EQU     X'F88E'     DISASSEMBLE WITHOUT PRINT
0000F8D0            INSTDSP EQU     X'F8D0'     DISASSEMBLE SINGLE INSTR.
                    *
0000002F            LENGTH  EQU     X'2F'       DISASSEMBLED INSTR LENGTH
0000003C            A1L     EQU     X'3C'       WORK BYTES FOR MONITOR
0000003D            A1H     EQU     X'3D'
00000040            A3L     EQU     X'40'
00000041            A3H     EQU     X'41'
00000044            A5L     EQU     X'44'
00000045            A5H     EQU     X'45'
0000003A            PCL     EQU     X'3A'       PC LOW FOR DISASSEMBLER
0000003B            PCH     EQU     X'3B'       ..TAKE A GUESS...
```

```
                    * ENTRY IS VIA CONTROL-Y AFTER
                    *  MOVING PROGRAM TO ITS NEW
                    *  LOCATION IN MEMORY. THE
                    *  VALUES FROM THE APPLE 'MOVE'
                    *  COMMAND WILL BE PRESENT IN
                    *  THE MONITOR WORK AREAS UPON
                    *  ENTRY TO RELOC8.
                    *
0300                        ORG     X'0300'     ORG TO PAGE 3
0300    A5 40       RELOC8  LDAZ    A3L         MOVE OLD BASE
0302    85 02               STAZ    OBL
0304    A5 41               LDAZ    A3H
0306    85 03               STAZ    OBH
                    *
0308    A5 3C               LDAZ    A1L         MOVE OLD END (+1)
030A    85 04               STAZ    OEL
030C    A5 3D               LDAZ    A1H
030E    85 05               STAZ    OEH
                    *
0310    A5 44               LDAZ    A5L         MOVE NEW BASE
0312    85 06               STAZ    NBL
0314    A5 45               LDAZ    A5H
0316    85 07               STAZ    NBH
```

```
                        *
                        * --- COMPUTE NEW END AND
                        *       RELOCATION BIAS.
                        *
0318      20 89 F6              JSR       SWEET16           GO TO SWEETIE
031B      23                    LD        NB
031C      B1                    SUB       OB                RELOCATION BIAS
031D      35                    ST        RB                 IS DIFFEREOCE
031E      22                    LD        OE
031F      B1                    SUB       OB                COMPUTE SIZE
0320      A3                    ADD       NB                ADD TO NEW BASE
0321      34                    ST        NE                 AND WE HAVE NEW END
0322      00                    RTN                         6502 MODE!


                        *
                        * SCAN THE PROGRAM FOR A 3-BYTE
                        *  INSTRUCTION. ANY OTHERS DON'T
                        *  HAVE TO BE RELOCATED. IF THE
                        *  ADDRESS IS OUTSIDE THE PROGRAM,
                        *  THEN WE CAN LEAVE IT ALONE.
                        *  OTHERWISE, UPDATE IT BY ADDING
                        *  THE RELOCATION BIAS.
                        *
0323      A0 00       GETINST   LDYIM     0                 DUMMY INDEX
0325      B1 06                 LDAIY     NBL               GET OPCODE
0327      20 8E F8              JSR       INSDS2            GET ITS LENGTH
032A      A5 2F                 LDAZ      LENGTH            CHECK LENGTH
032C      C9 02                 CMPIM     2                 3 BYTES?
032E      D0 24                 BNE       NXTINST           =>NOPE. SKIP IT.
                        *
                        * IF THE ADDRESS IS WITHIN THE
                        *  PROGRAM, RELOCATE IT.
                        *
0330      20 89 F6              JSR       SWEET16           HI, SWEETIE!
0333      E3                    INR       NB                BUMP TO ADDRESS
0334      63                    LDD       NB                GET BOTH BYTES
0335      D1                    CPR       OB                >= OLD BASE?
0336      02 2A                 BNC       NXT1              =>LOWER. NO CHANGE.
0338      D2                    CPR       OE                <= OLD END?
0339      03 27                 BC        NXT1              =>HIGHER. NO CHANGE.
                        *
                        * ADD RELOCATION BIAS.
                        *
033B      A5                    ADD       RB                ADD BIAS
033C      F3                    DCR       NB                BACK UP TO
033D      F3                    DCR       NB                 ADDRESS AGAIN
033E      73                    STD       NB                STUFF BACK THERE
                        *
                        * --- ANNOUNCE THE CHANGE --- *
                        *
033F      23                    LD        NB                BACK UP POINTER
0340      F0                    DCR       AC                 TO OPCODE
0341      F0                    DCR                           FOR THE
0342      F0                    DCR                            DISASSEMBLER
0343      00                    RTN                         BACK TO 6502 MODE
0344      A5 00                 LDAZ      ACL               MOVE POINTER
0346      85 3A                 STAZ      PCL                TO PCH/PCL
0348      A5 01                 LDAZ      ACH                FOR THE
034A      85 3B                 STAZ      PCH                 DISASSEMBLER
034C      20 D0 F8              JSR       INSTDSP           PRINT MODIFIED INSTR.
034F      20 89 F6              JSR       SWEET16           RE-ENTER SWEET16 TO
0352      01 0E                 BR        NXT1               CONTINUE...
```

```
                        *
                        * WE'VE GOT A 1 OR 2 BYTE
                        *  INSTRUCTION. UPDATE THE
                        *  NB POINTER TO THE NEXT
                        *  INSTRUCTION.
                        *
0354      18            NXTINST  CLC
0355      69 01                  ADCIM    1          UPDATE LENGTH: 1/2/3
0357      85 00                  STAZ     ACL        GET LENGTH
0359      A9 00                  LDAIM    0          HI=0
035B      85 01                  STAZ     ACH
035D      20 89 F6               JSR      SWEET16    BACK TO SWEET16
0360      A3                     ADD      NB         BUMP IT
0361      33                     ST       NB         PUT BACK THERE
                        *
                        * CHECK TO SEE IF WE'RE DONE
                        *  WITH THE PROGRAM YET.
                        *
0362      23            NXT1     LD       NB         GET CURRENT ADDRESS
0363      D4                     CPR      NE         OVER THE END?
0364      03 04                  BC       DONE       =>YUP. ALL DONE!
0366      00                     RTN                 =>NO. BACK TO THE
0367      B8                     CLV                    6502 MODE FOR
0368      50 B9                  BVC      GETINST     MORE WORK!
                        *
                        * ALL DONE. EXIT TO MONITOR.
                        *
036A      00            DONE     RTN                 6502 MODE, PLEASE!
036B      60                     RTS                 BACK TO MONITOR!
                        *
                        *
03F8                            ORG      X'03F8'    CONTROL-Y ENTRY
03F8      4C 00 03              JMP      RELOC8      ROLL STONE, GATHER MOSS...
                        *
                               END
```

PET

A MEMORY TEST PROGRAM FOR THE COMMODORE PET

Michael J. McCann
28 Ravenswood Terrace
Cheektowaga, NY 14225

It would be useful and convenient to be able to test PET's memory with a testing program rather than sending the machine back to Commodore for service. Towards this end I have written a memory test program in Commodore BASIC for the PET. The program is well commented, and should be self documenting. (see listing)

Since the program occupies the lowest 4K of PET's memory, use of the program will require that the lowest 4K of memory be operating normally. The amount of time required to run this program rapidly increases as the number of bytes under test is increased (see Figure 1.)

Testing large blocks of memory results in more rigorous testing at the expense of time. Therefore, when using this program the user will have to make a decision regarding rigor vs. time. As a bare minimum, I would suggest testing 100 bytes at a time.

In closing I would suggest that you get this program up and running before you have a problem. It may prove difficult to get a new program working when you have a major system problem.

```
10 REM MEMORY TEST PROGRAM FOR THE COMMODORE PET
20 REM PROGRAM WILL RUN ON 8K PET
30 REM BY MICHAEL J MCCANN
40 PRINT CHR$(147):EE=0:I=0
50 INPUT "START ADDRESS"; SA
60 IF SA<4097 OR SA>65535 GOTO 50
70 INPUT "STOP ADDRESS"; SP
80 IF ST>65535 OR SP<SA GOTO 70
90 PRINT CHR$(147):PRINT:PRINT
100 PRINT TAB(5)"WORKING"
105 PRINT:PRINT"FAULT IN ADDRESS:";
110 REM MEMORY ACCESS AND LOGIC CIRCUITRY TEST
120 REM WRITE ALL 0
130 FOR A=SA TO SP
140 POKE A,0
150 NEXT
160 REM CHECK FOR CORRECTNESS (=0)
170 FOR A=SA TO SP
180 IF PEEK(A)<>0 THEN EE=1:GOSUB 800
190 NEXT
200 REM WRITE ALL 255
210 FOR A=SA TO SP
220 POKE A,255
230 NEXT
240 REM CHECK FOR CORRECTNESS(=255)
250 FOR A=SA TO SP
260 IF PEEK(A)<>255 THEN EE=1:GOSUB 800
270 NEXT
280 REM BEAT TESTS
290 REM WRITE ALL 0
300 FOR A=SA TO SP
310 POKE A,0
320 NEXT
330 REM BEAT ONE ADDRESS WITH 255
335 AD=SA+I
340 POKE AD,255
350 POKE AD,255
360 POKE AD,255
370 POKE AD,255
380 POKE AD,255
390 REM CHECK ALL FOR 0 EXCEPT THE ADDRESS
    BEAT WITH 255
400 FOR A=SA TO SP
410 IF A=AD GOTO 430
420 IF PEEK(A)<>0 THEN EE=1:GOSUB 800
430 NEXT
440 IF AD=SP+1 THEN POKE AD,0: I=I+1: GOTO 335
450 I=0
460 REM WRITE ALL 255
470 FOR A=SA TO SP
480 POKE A,255
490 NEXT
500 REM BEAT ONE ADDRESS WITH 0
505 AD=SA+I
510 POKE AD,0
520 POKE AD,0
530 POKE AD,0
540 POKE AD,0
550 POKE AD,0
560 REM CHECK ALL FOR 255 EXCEPT THE ADDRESS
    BEAT WITH 0
570 FOR A=SA TO SP
580 IF A=AD GOTO 600
590 IF PEEK(A)<>255 THEN EE=1:GOSUB 800
600 NEXT
610 IF AD<>SP+1 THEN I=I+1:POKE AD,255:GOTO 505
620 REM ADDRESSING TEST
630 REM WRITE CONSECUTIVE INTEGERS (0-255) IN
    ALL LOCATIONS UNDER TEST
640 I=0
650 FOR A=SA TO SP
660 IF I=256 THEN I=0
670 POKE A,I
680 I=I+1
690 NEXT
700 REM CHECK FOR CORRECTNESS
705 I=0
710 FOR A=SA TO SP
720 IF I=256 THEN I=0
730 IF PEEK(A)<>I THEN EE=1:GOSUB 800
740 I=I+1
750 NEXT
760 PRINT
770 IF EE=0 THEN PRINT" NO MEMORY PROBLEMS DE-
    TECTED"
780 END
800 PRINT A;
810 RETURN
```

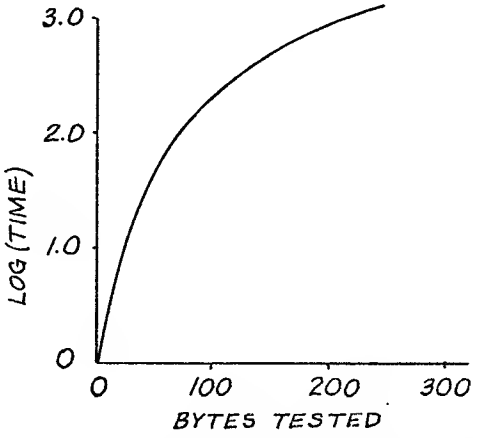


Figure 1. Graph of Log(Time Required) vs. Number of Bytes Tested. (Time in Seconds)

PEEKING AT PET'S BASIC

Harvey B. Herman
Chemistry Department, U. of N. Carolina
Greensboro, NC 27412

Commodore, for reasons best known to them, has seen fit to prevent users from PEEKing at PET's ROM located, 8K BASIC. If you try to run a program that says, PRINT PEEK (49152), the answer returned will be zero instead of the actual instruction or data in decimal. Disassemblers written in BASIC will therefore not work properly if they use the PEEK command and try to disassemble 8K BASIC (decimal locations 49152 to 57520). I was curious to see how the PET's 8K BASIC was implemented and decided to write a machine language program which circumvents the restriction.

A listing of the above program which I have called MEMPEEK follows. It is decimal 22 bytes long, relocatable, and can be stored into any convenient area of memory. I have chosen to use the area devoted to the second cassette buffer starting at hex 33A. As long as the second cassette is not used the program should remain inviolate until the PET is turned off. Storing the program in memory is trivial if a machine language monitor is available. Otherwise convert the hex values to decimal and manually poke the values into memory. As of this writing, Commodore's free, long-awaited, TIM-like monitor has not arrived but I continue to hope.

MEMPEEK utilizes the user function (USR) which jumps to the location stored in memory locations 1 and 2. If MEMPEEK is stored in the second cassette buffer (hex 33A) initialize locations 1 and 2 to decimal 58 and 3 respectively. MEMPEEK was written so that the user function returns the decimal value of the instruction given by its argument (address). For example, if you want to peek at an address less than decimal 32768 (not part of the BASIC ROMs) use in your program Y=USR (address), where address is the location of interest and the value of Y is set to the instruction at that address. Since the argument of the user function is limited to +32767, use address -65536 for addresses larger than 32768. Thus to look at locations in the BASIC ROMs (all above 32768 and where MEMPEEK is particularly useful) use Y=USR (address -65536). It is not possible to look at location 32768 (the start of the screen memory) with this program but this should prove no handicap as PEEK could be used.

MEMPEEK takes advantage of two subroutines in the PET operating system. The first (located at hex D0A7) takes the argument (address) in the floating point accumulator (conveniently placed there by the user function) and converts it into a two byte integer stored at hex B3 and B4. Since I choose to use an indirect indexed instruction to find the desired instruction the order of the two bytes at hex B3 (MSB) and B4 (LSB) need to be reversed. The second subroutine at hex D278 converts a 2 byte integer representing the instruction from the accumulator (MSB) and the Y register (LSB) to floating point form and stores it in the floating point accumulator. This value, the instruction, is returned to BASIC as the result of the user function.

The program, MEMPEEK, is fairly simple but would be unnessary if the arbitrary restriction on PEEKing at BASIC was removed. The restriction makes no sense to me as even a relatively inexperienced machine language programmer (myself) was able to get around it. This type of program would of course not be difficult for competitors of Commodore to write. I wrote this program for the fun of it, to try to understand how BASIC works and in the hope others will find it useful. Furthermore, I hope I can discourage other manufacturers like Commodore from trying to keep hobbyists from a real understanding of their software by arbitrary restrictions.

MEMPEEK Program

```
033A             1 *=$33A
033A  20A7D0     2 JSR $D0A7 ; convert to integer
033D  A6B3       3 LDX $B3   ; interchange -
033F  A4B4       4 LDY $B4   ; $B3 and $B4
0341  86B4       5 STX $B4
0343  84B3       5 STY $B3
0345  A200       7 LDX #0    ; initialize index
0347  A1B3       8 LDA ($B3,X); find instruction
0349  A8         9 TAY
034A  A900      10 LDA #0
034C  2078D2    11 JSR $D278 ; convert to floating
034F  60        12 RTS       ; return to BASIC
0350            13 END
```

Gary A. Creighton
625 Orange Street, No. 43
New Haven, CT 06510

I am writing this article because I'm tired of seeing the same rehash of pseudo-facts being repeated about the PET. If I read one more time about the small keyboard or the RND function not working correctly...! As you will see, the 2001 has an extremely well designed Interpreter which can be used effectively as subroutines either from the SYS command, or the USR command. Parameter passing will be revealed as an easy operation, and returning USR with a value is just as simple. The RND function may be substituted with a twelve byte USR program to make it completely random and non-repeating (as it stands, it repeats every 24084 times through) and I will show the use of negative arguments. Unfortunately, RND(0) was apparently a mis-calculation on Microsoft's part. They figured that ROM empty locations would turn out to be more random than the end product shows. They load non-existent memory locations into the RND store area (218-222) thus causing a resulting RND value which fluctuates between a few different values. When ROM is finally installed in that area (36932) the RND(0) will have the dubious quality of being some fixed number.

RND FUNCTION USE

The RND function may be set at any time to execute a known series of RND #'s by using a known negative argument just before RND with a positive one. The ability to have available a known list of random numbers is very important in a lot of sciences.

```
10 R=RND(-1)
20 FOR X=1 TO 5
30 PRINT INT(1000*RND(1)+1),
40 NEXT X
```

Gives the sequence: 736, 355, 748, 166,629

Since RND(-low#) gives such a small value, use a negative argument in the range (-1 E10 to -1 E30) if you need one repeatable RND number with a useful value, e.g., RND(-1 E20)= .811675238.

Concerning the true random nature of RND and it's ability to act randomly at all times; time must be combined with RND. This is possible with a RANDOMIZE subroutine or faster still, redoing RND(+) with a USR routine.

```
10000 REM (RANDOMIZE)
10010 R1=PEEK(514) : R2=PEEK(517)
10020 POKE 220, R1 : POKE 221, R2
10030 RETURN
```

This routine may be used at program initialization and as the program halts for an INPUT. It will start a new sequence of RND numbers whenever called.

When the computer does a sequence without intervention, the following USR program is suggested which will return a truly random number quickly; without repeating.

```
10 REM (TRUE RND USING USR FUNCTION)
20 POKE 134,214 : POKE 135,31 : CLR
30 FOR X=8150 TO 8165
40 READ BYTE : POKE X, BYTE
50 NEXT X
60 DATA 173,2,2,133,220,173,5,2,133,221,76
65 DATA 69,223,0,0,0
70 POKE 1, 214 : POKE 2, 31
```

MACHINE LANGUAGE STORING IN BASIC

When using machine language, always precede storing by setting up BASIC's upper boundary. This is done by:

```
POKE 134, ITEM : POKE 135, PAGE : CLR
```

e.g. `POKE 134, 0 : POKE 135, 25 : CLR`
sets upper boundary to 6400 and BASIC use will be confined to 1024 to 6399 unless reset or turned off.

You can use the following program for storing decimal. Changing INDEX to 10000 to appropriate position and typing in DATA lines in 100 to 9997.

```
0 REM ("MACHINE STORE")
1 REM WRITTEN BY GARY A. CREIGHTON, JULY 78
2 REM ( SET INDEX=ORIGIN IN LINE 10000 )
3:
15 REM FIX UPPER STRING BOUNDARY
20 GOSUB 10000
25 X=INDEX / 256
30 PAGE=INT(X)
35 ITEM=(X-PAGE)* 256
40 POKE 134, ITEM
45 POKE 135, PAGE
50 CLR
55 :
60 REM  LOAD MACHINE LANGUAGE
65 GOSUB 10000 : LOC=INDEX
70 READ BYTE : IF BYTE<0 THEN END
75 POKE LOC, BYTE
80 LOC=LOC+1 : GOTO 70
85 :
90 REM MACHINE LANGUAGE DATA
100 DATA
     .
     .
9997 DATA
9998 DATA 0,0,0,-1
9999 :
10000 INDEX=(START OF MACHINE LANGUAGE)
10010 RETURN
```

USR PARAMETER PASSING

The following are parameter passing rules for the USR function and should be added to the "MACHINE STORE" program.

```
0 REM ("USR(0 TO 255)")
46 POKE 1, ITEM
48 POKE 2, PAGE
100 REM (USR INPUT 0-255; OUTPUT 0-255)
110 DATA 32,121,214   : REM JSR 54905
120 DATA (Your program using input value)
     .
     .
5000 DATA (Setup output value in Accum.)
5010 DATA 76,245,214   : REM JMP 55029
10000 INDEX 6400
```

OR

```
0 REM ("USR(0 TO 65535)")
46 POKE 1, ITEM
48 POKE 2, PAGE
100 REM (USR INPUT 0-65535;OUTPUT 0-65535)
110 DATA 32,208,214  : REM JSR 54992
    (Note: Check if 0-65535. RTS with:
           Y and M(8)= ITEM
           A and M(9)= PAGE
120 DATA (Your program using 2 byte passed
         value)
     .
     .
5000 DATA (Setup output vlaue ITEM in Y;
          PAGE in A)
5010 DATA 132,178     : REM STYZ 178
5020 DATA 133,177     : REM STAZ 177
5030 DATA 162,144     : REM LDXIM 144
5040 DATA 56          : REM SEC
5050 DATA 76,27,219   : REM JMP 56091
     (Setup output value and RTS)
```

The input parameter may be any complex expression and you can of course:

input 0-255 and output 0-65535, or
input 0-65535 and output 0-255.

SAVE MACHINE LANGUAGE AND LOAD DIRECTLY

The reason for the 0,0,0 at the end of the preceding machine language programs is that the saving routine described next SAVES machine language until 0,0,0 or an ERROR is printed. After it has been saved in this way, it may be LOADED and VERIFIED with little effort.

Add to "MACHINE STORE" program (all assembly is in decimal).

```
        0 REM ("SAVEM")
        100 REM ERAM=31 (or last page of RAM on your PET)
        110 DATA 32,200,0    : REM  JSR   200    check if : or end of line
        120 DATA 208,3       : REM  BNE   OVER
        130 DATA 76,158,246  : REM  JMP   63134  jump 'SAVE' if SYS 8000 only
OVER    140 DATA 32,17,206   : REM  JSR   52753  check if ','
        150 DATA 32,164,204  : REM  JSR   52388  analyze arithmetical argument
        160 DATA 32,208,214  : REM  JSR   54992  check if 0-65535
        170 DATA 132,247     : REM  SYTZ  247    'save from' item
        180 DATA 133,248     : REM  STAZ  248    'save from' page
        190 DATA 170         : REM  TAX
        200 DATA 152         : REM  TYA
        210 DATA 208,1       : REM  BNE   OVR2
        220 DATA 202         : REM  DEX
OVR2    230 DATA 136         : REM  DEY          back up 1
        240 DATA 132,80      : REM  STYZ  80     initialize CHK pointer item
        250 DATA 134,81      : REM  STXZ  81     initialize CHK pointer page
        260 DATA 169,173     : REM  LDAIM 173
        270 DATA 133,79      : REM  STAZ  79     LDA instruction in 0079
        280 DATA 169,96      : REM  LDAIM 96
        290 DATA 133,82      : REM  STAZ  82     RTS instruction in 82
        300 DATA 32,200,0    : REM  JSR   200
        310 DATA 201,44      : REM  CMPIM 44     check if ',' before filename
        320 DATA 208,3       : REM  BNE   OVR3
        330 DATA 32,194,0    : REM  JSR   194    move code pointer over ','
OVR3    340 DATA 32,51,244   : REM  JSR   62515  get options for "SAVE'
AGAIN   350 DATA 230,80      : REM  INCZ  80
        360 DATA 208,2       : REM  BNE   OVR4
        370 DATA 230,81      : REM  INCZ  81     add 1 to CHK pointer
OVR4    380 DATA 32,79,0     : REM  JSR   79     look at next CHK code
        390 DATA 208,27      : REM  BNE   CHEND
        400 DATA 160,1       : REM  LDYIM 1      check for 0,0,0
        410 DATA 177,80      : REM  LDAIY 80
        420 DATA 208,21      : REM  BNE   CHEND
        430 DATA 200         : REM  INY
        440 DATA 177,80      : REM  LDAIY 80
        450 DATA 208,16      : REM  BNE   CHEND
        460 DATA             24     : REM CLC
        470 DATA             165,80 : REM LDAZ   80
        480 DATA             105,4  : REM ADCIM  4
        490 DATA             13
        460 DATA 24          : REM  CLC
        470 DATA 165,80      : REM  LDAZ  80
        480 DATA 105,4       : REM  ADCIM 4
        490 DATA 133,299     : REM  STAZ  229    'save to' item
        500 DATA 165,81      : REM  LDAZ  81
        510 DATA 105,0       : REM  ADCIM 0
        520 DATA 133,230     : REM  STAZ  230    'save to' page
        530 DATA 76,177,246  : REM  JMP   63153  complete 'SAVE'
```

```
CHEND   540 DATA 165,81      : REM  LDAZ  81
        550 DATA 201,31      : REM  CMPIM ERAM
        560 DATA 240,10      : REM  BEQ   CHKNF  check: 'not found' if last
        570 DATA 144,210     : REM  BCC   AGAIN  look at next if less than
        580 DATA 32,184,31   : REM  JSR   END
        590 DATA 162,85      : REM  LDXIM 85
        600 DATA 76,108,195  : REM  JMP   70028  ("?END) NOT FOUND ERROR"
CHKNF   610 DATA 165,80      : REM  LDAZ  80
        620 DATA 201,253     : REM  CMPIM 253
        630 DATA 144,196     : REM  BCC   AGAIN  again if enough room
        640 DATA 32,184,31   : REM  JRS   END
        650 DATA 160,40      : REM  LDYIM 40
        660 DATA 76,133,245  : REM  JMP   62853  ("?END) NOT FOUND ERROR"
END     670 DATA 169,13      : REM  LDAIM 13
        680 DATA 32,234,227  : REM  JSR   58346
        690 DATA 169,63      : REM  LDAIM 63
        700 DATA 32,234,227  : REM  JSR   58346
        710 DATA 169,69      : REM  LDAIM 69
        720 DATA 32,234,227  : REM  JSR   58346
        730 DATA 169,78      : REM  LDAIM 78
        740 DATA 32,234,227  : REM  JSR   58346
        750 DATA 169,68      : REM  LDAIM 68
        760 DATA 32,234,227  : REM  JSR   58346  "?END"
        770 DATA 96          : REM  RTS
        780 REM (FORMAT: SYS 8000,INDEX,"FILENAME",DEVICE#,I/O OPTION)
```

After typing and saving normally, type RUN when READY. Save "SAVEM" using itself to save itself by typing:

SYS 8000,8000, "SAVE(SYS 8000)"

when READY., REWIND TAPE #1 and type:

VERIFY "SAVE(SYS 8000)"

MACHINE LANGUAGE LOAD PROCEDURE

After SAVEing machine language, you have the capability of LOADing directly if you follow these rules.

Loading machine language before BASIC program:

```
LOAD "machine language name"
NEW
A=PEEK(247) :B=PEEK(248)
POKE 134,A  :POKE 135,B
POKE 1,A    :POKE 2,B (only if USR, not SYS)
CLR
```

Then LOAD BASIC Program.

Loading machine language from BASIC program:

```
0 IF OK THEN RUN 6
1 OK=-1 : PRINT "PRESS REWIND ON TAPE #1"
2 WAIT 519,4,4 : REM wait til stop if play down but not motor
3 WAIT 59411,8,8 : REM wait til key on cassette pushed
4 WAIT 59411,8 : REM wait til stop on cassette pushed
5 LOAD "machine language name"
6 A=PEEK(247) : B=PEEK(248)
7 POKE 134,A  : POKE 135,B
8 POKE 1,A    : POKE 2,B : REM (only if USR, not SYS)
9 CLR
10 REM (BEGIN BASIC PROGRAM, MACHINE LANGUAGE LOADED)
```

HOW GOES YOUR ROM TODAY?

Harvey B. Herman
Chemistry Department
University of North Carolina-Greensboro
Greensboro, North Carolina 27412

Everytime I turn on my KIM-system or PET Personal Computer I keep my fingers crossed that everything works. So far I have been "lucky" and the few failures were patently obvious. However, I have been concerned about the possibility of subtle errors appearing which, while not obvious, will still cause programs to print garbage out without my having inputted garbage. To ease my troubled mind, I wrote an assembly language program which computes a checksum byte from the data in a specified area of memory. The 6502 programs, which I named CHECK, can be used to check data in both ROMs and RAMs for erroneous bits.

The program for a KIM system is shown in Figure 1. It can be entered into memory with the KIM monitor program or an assembler. With a few minor changes, which I believe are obvious by looking at the code, it can be placed practically anywhere in memory. The program requires four zero page locations to be initialized to the starting and ending locations of the specified area. I used locations hex E1, E2 and E3, E4 respectively (low byte first) as these were the first free page zero locations in Microsoft 8K BASIC. The reader may wish to change these locations if it interfers with other programs that are frequently used. The KIM CHECK program ends with a BRK (break) instruction and will not operate properly unless two locations, hex 17FE, 17FF, are initialized to 00, 1C, respectively. The BRK instruction, when executed will then jump to the start of the KIM monitor and among other things, print the value saved in location hex 31D - the calculated checksum. Initialization and executation of this program can be done with the KIM monitor. The checksum bytes which I calculated for two different KIM system ROMs are shown in Table 1.

Several changes are necessary that allow a similar program to work on Commodore's PET computer. The modified program is shown in figure 2 and is a listing from a cross assembly done on the KIM system. The values could be placed in memory with a monitor program, if available, or as I did, poked into memory from a BASIC program. The latter approach requires a conversion from hex to decimal before using the POKE command. Again, as before, four locations in page zero need to be initialized. Part of the area reserved for the second cassette buffer was used for the program (hex 33A-371) and four locations (hex 53-56) in the keyboard buffer were used for the page zero locations representing the starting and ending locations of the area to be checked. The PET CHECK program is designed to be run from BASIC. A call to the USR (user) function, ?USR(0), jumps to the checksum program and returns the checksum value. The program has two entry points. It can be used to calculate checksums (see Table 1) for the BASIC interpreter and/or the operating system (both are in ROM) or BASIC programs which have just been loaded or saved. The latter use somewhat obviates the need to use the VERIFY tape command after a load. This can save considerable time particularly if long programs are loaded. Alternate entry points are specified by POKEing locations 1 and 2 to decimal 58 and 3 for program checks and to decimal 82 and 3 for ROM checks, respectively. The starting and ending locations in page zero are automatically set by the program for program checks but must be specified for ROM checks.

Further details on the use of each program is shown in Table 2. The checksums calculated are the exclusive OR of all the bytes between the starting and ending addresses, inclusively. Changing as little as one bit in the sequence will give a different value for the checksum. There is a finite probability that when extensive errors are encountered the checksum calculated would fortuitously be the same, since only 256 different 8 bit checksums are possible. However, in that case the errors would probably not be subtle and you would not be fooled. Whenever the checksums for the ROMs change it would be prudent also to run a diagnostic test on the 6502 MPU before blaming the ROM. Since programs like that are sadly lacking I will leave it as an exercise for the reader. A program and article to that effect would be greatly appreciated by the author for one, and I believe most of 6502 personal computing fraternity.

KIM ROMs (Serial numbers 1988 and 6931)

Locations (Hex)	Checksum (Hex)
1800-1BFF	F5
1C00-1FFF	F8
1800-1FFF	0D

KIM CHECK Program. Example for 1800-1FFF. After placing program from Figure 1 into memory

```
KIM
17FE  0.
17FF  1C.          0300  AD  G
E1    0.           KIM
E2    18.          031D  (CHECKSUM)
E3    FF.
E4    1F.
```

PET ROMs (Serial numbers 10252 & 20549)

Locations (Hex)	Loc.(Dec., Inv.)	Check
C000-CFFF	0,192-255,207	189
D000-DFFF	0,208-255,223	87
E000-E777	0,224-119,231	26
F000-FFFF	0,240-255,255	92

PET CHECK Program. After poking program from Figure 2 into memory

Program Checks	ROM Checks
POKE 1,58	(Example for C000-CFFF)
POKE 2,3	
LOAD "program name"	POKE 1,82
or	POKE 2,3
SAVE "program name"	POKE 83,0
?USR (0)	POKE 84,192
(checksum returned depends on program)	POKE 85,255
	POKE 86,207
	?USR (0)
	189 (Checksum returned)

```
033A             1 ;      KIM CHECKSUM PRØGRAM
033A             2 ;      HARVEY B. HERMAN
033A             3 ;      INITIALIZE $17FE/FF
033A             4 ;      TØ Ø/1C SØ BRK WØRKS.
00E1             5        *=$E1
00E1 0000        6 START  .WØRD Ø
00E3 0000        7 END    .WØRD Ø
0300             8        *=$300
0300             9 ;      ENTER HERE FØR
0300            10 ;      CALCULATIØN ØF
0300            11 ;      CHECKSUM BETWEEN
0300            12 ;      START AND END.
0300            13 ;      ANS DISPLAYED LØC 315
0300 A000       14        LDY #Ø
0302 B1E1       15        LDA (START),Y
0304 E6E1       16 LØØP   INC START
0306 D002       17        BNE CHECK
0308 E6E2       18        INC START+1
030A 51E1       19 CHECK  EØR (START),Y
030C A6E4       20        LDX END+1
030E E4E2       21        CPX START+1
0310 D0F2       22        BNE LØØP
0312 A6E3       23        LDX END
0314 E4E1       24        CPX START
0316 D0EC       25        BNE LØØP
0318 8D1D03     26        STA *+5
031B 00         27        BRK
031C            28        .END
```

Figure 1
KIM Checksum Program.

```
033A             1 ;      PET CHECKSUM PRØGRAM
033A             2 ;      HARVEY B. HERMAN
0053             3        START=$53
0055             4        END=$55
033A             5 .      *=$33A
033A             6 ;      ENTER HERE TØ CHECK
033A             7 ;      BASIC PRØGRAMS AFTER
033A             8 ;      LØAD ØR SAVE.
033A A900        9 PRØG   LDA #0
033C 8553       10        STA START
033E A904       11        LDA #4
0340 8554       12        STA START+1
0342 A5E6       13        LDA $E6
0344 8556       14        STA END+1
0346 A5E5       15        LDA $E5
0348 38         16        SEC
0349 ED7103     17        SBC TWØ
034C B002       18        BCS SKIP
034E C656       19        DEC END+1
0350 8555       20 SKIP   STA END
0352            21 ;      ENTER HERE TØ CHECK
0352            22 ;      ANY LØCATIØNS IN
0352            23 ;      MEMØRY. INITIALIZE
0352            24 ;      $53-$56 FIRST.
0352 A000       25 RØM    LDY #0
0354 B153       26        LDA (START),Y
0356 E653       27 LØØP   INC START
0358 D002       28        BNE CHECK
035A E654       29        INC START+1
035C 5153       30 CHECK  EØR (START),Y
035E A656       31        LDX END+1
0360 E454       32        CPX START+1
0362 D0F2       33        BNE LØØP
0364 A655       34        LDX END
0366 E453       35        CPX START
0368 D0EC       36        BNE LØØP
036A A8         37        TAY
036B A900       38        LDA #0
036D 2078D2     39        JSR $D278
0370 60         40        RTS
0371 02         41 TWØ    .BYTE 2
0372            42        .END
```

Figure 2
PET Checksum Program

HIGH-RESOLUTION PLOTTING FOR THE PET

John R. Sherburne
206 Goddard
White Sands Missile Range, NM 88002

The PET Machine Language Monitor gives PET users a greatly expanded ability to devleop and use assembly language programs. While early buyers of PET have had to wait a while for the Monitor, the ability to save and load machine language programs directly to and from cassette is well worth the wait. Access to machine language has always been available through the POKE command, but translating op codes and addresses from hex to decimal and back is tedious. Also, the need to load a program via another BASIC program or via the keyboard is wasteful and time-consuming. PET's Monitor allows an assembly language program to be saved and loaded as easily as the BASIC program. Better yet, an assembly language program can be written to reside in an unused section of memory such as the second cassette buffer. A BASIC program can then be loaded in the usual manner and can use the machine language program as a subroutine.

One way that the use of a resident machine language routine can be a big help is in implementing high-resolution plotting on the PET. High-resoltuion plotting, in effect, expands PET's 40 × 25 character display to 80 × 50. To do so, each character is divided into quarter characters. The four basic quarter characters are displayed by pressing "SHIFT" and "," or ";" or "<" or ">". There are a total of sixteen possible combinations of these four quarter characters which can be used to produce a high-resolution plot. The process of producing such a plot in BASIC, however, is complex and slow. A machine language subroutine, on the other hand, can make the plotting process quite simple. For example, the Lissajous figure in * Figure 1 was plotted with this program:

```
10 POKE 1,58:POKE 2,3:PRINT (clr)"
20 DELTA=2*π/900
30 P=3:Q=4
40 FOR I=0 TO 900
50 THETA=DELTA*I
60 X=INT(39.5+38*COS(P*THETA))
70 Y=INT(25.5+24*SIN(Q*THETA))
80 POKE 81,X:POKE 82,Y:A=USR(0)
90 NEXT I
100 GET A$:IF A$="" THEN 100
```

The machine language routine is called in line 80 with the USR command after first POKEing the X and Y coordinates to be plotted in memory locations 81 and 82, respectively. The values of P and Q in line 30 determine the shape of the figure. The machine language plotting routine used by the program is listed below. The procedures for using it are:

LOADING - The program is initially loaded into the second cassette buffer beginning in location $033A using the Monitor. The program is saved on cassette with the command: .S,01,HI-RES,033A,03CA. The value $03CA is the endng address plus one. Once saved, the program can be reloaded into the cassette buffer with the normal command: LOAD"HI-RES".

BASIC INTERFACE - With HI-RES loaded, the BASIC driver program can be loaded from cassette using normal procedures or the "NEW" command can be given and a new BASIC program entered from the keyboard. Before HI-RES can be called, the starting address, $033A, must be entered in memory locations 0001 and 0002. This was done in line 10 of the program above. HI-RES can now be called by the USR command. Before each call, the X and Y coordinates must be POKED into decimal addresses 81 and 82, respectively. Valid coordinate values run from 0 to 79 in the X direction and from 0 to 49 in the Y direction. Position 0, 0 is in the upper left-hand corner of the screen.

OTHER - If zero is used as the argument of the USR command, the plotting routine will overwrite any character already on the screen. If a value other than zero is used any non-plot character already on the screen will be left there. Thus axes and text can be preprinted on the screen and a graph later plotted without distrubing the preprinted data.

RECREATIONAL GRAPHICS FOR PET

There are probably a lot of practical uses for the PET high-resolution graphics program described above but I haven't had time to find them yet. Instead, I have spent countless hours in front of the display watching PET draw intriguing designs for which there is relatively little practical purpose. My addiction started simply enough. To test the HI-RES plotting routine, I wrote a program to draw an elipse using the formula: X=P*COS(Θ); Y=Q*SIN(Θ). Pleased with the result, I added a FOR loop to vary the values of P and Q and produced the family of elipses shown in Figure 1. I didn't realize it but I had embarked on a project which would take every free moment for the next two weeks.

The next step was to modify the formula so that a flower rather than an elipse was produced. The new formula was:

X=R*COS(θ); Y=R*SIN(θ) where R=SIN(N*θ) If N is odd, a flower with N leaves is produced; if N is even, the flower will have 2N leaves. Figure 2A is an eight leaved flower using the formula R=SIN(4*θ). Figure 2B uses an alternate forumul: R=COS(4*θ). As with the ellipse, the next step was to produce a family of flowers (Figure 2C) by adding a FOR loop to vary the size of the flower and to alternate between the two formulas.

By now I was completely hooked. I dug into a dusty book of mathematical formulas and found two rather obscure figures, the epicycloid and hypocycloid. Best known from the toy "Spirograph", the epicycloid is formed by tracing the path of a point on the circumference of a circle as it is rolled around the outside of a second circle. The hypocycloid is formed when one circle is rolled around the inside of the other. The formulas are:

Epicycloid:

```
X=(P+Q)*COS(AN)+Q*COS(P+Q)*AN/Q)
Y=(P+Q)*SIN(AN)+Q*SIN(P+Q)*AN/Q)
```

Hypocycloid:

```
X=(P-Q)*COS(AN)+Q*COS(P-Q)*AN/Q)
Y=(P-Q)*SIN(AN)-Q*SIN(P-Q)*AN/Q)
```

*Note: Figure 1 on cover

In both formulas P represents the radius of the stationary circle and Q the radius of the rolling circle. A typical epicycloid is shown in Figure 3. To plot these more complex figures a minor technical problem had to be solved. Many of the larger "cycloids" require more than one revolution of the rolling circle around the stationary circle. To avoid either stopping too soon or running too long, I had to add a routine to compute the number of revolutions required for the full figure. Since the rolling circle makes P/Q Revolutions in one circuit of the stationary circle, a complete figure is made when the rolling circle turns the number of times equal to the first integer multiple of P/Q. That multiple, N, times 2 is the number of points or cusps in the cycloid. For convenience I print the number of cusps in the corner of the display. An eight cusp hypocycloid is shown in Figure 4. With both types of cycloid P and Q can be varied to produce a variety of figures. To avoid creating a figure too large to display, P must be ≤ 24 for a hypocycloid and $P+2*Q \leq 24$ for an epicycloid.

As a final fillip, a third parameter can be added to the cycloid programs. Rather than trace a point on the circumference of the rolling circle, a point at a distance R from the center of the circle is traced. The value of R can be larger or smaller than Q. If R is larger than Q the formulas for determining the largest figure which the display can accomodate are: epicycloid, $P+Q+R \leq 24$; hypocycloid, $P+R-Q \leq 24$.

HI-RESOLUTION

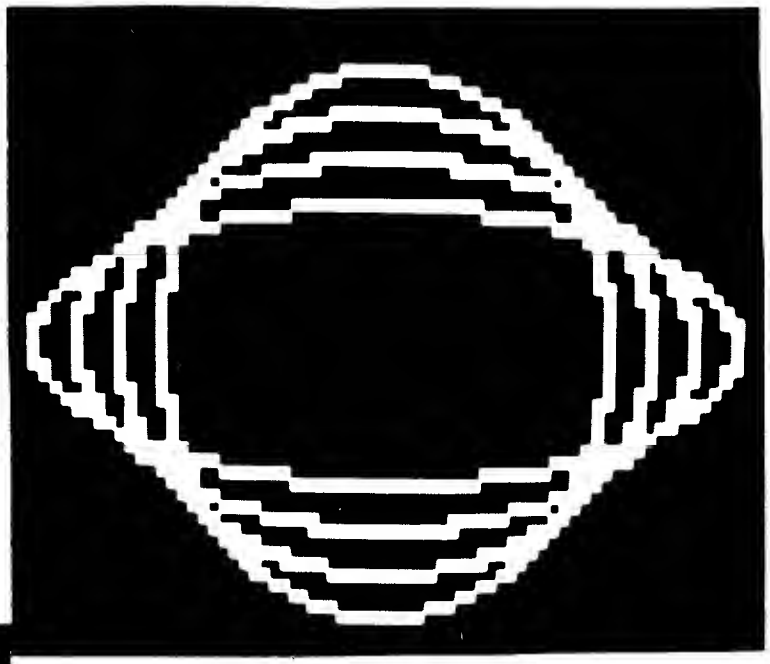
BY JOHN R. SHERBURNE
FEBRUARY 1979

```
033A                    ORG   $033A

033A A9 00      START   LDAIM $00     INITIALIZE
033C 85 53              STA   $0053
033E 85 56              STA   $0056
0340 38                 SEC
0341 A5 51              LDA   $0051
0343 E9 4F              SBCIM $4F     CHECK FOR VALID X
0345 30 03              BMI   CHECK
0347 E6 54              INC   $0054
0349 60                 RTS

034A 38         CHECK   SEC           CHECK FOR VALID Y
034B A5 52              LDA   $0052
034D E9 31              SBCIM $31
034F 30 03              BMI   HALF
0351 E6 55              INC   $0055
0353 60                 RTS

0354 46 51      HALF    LSR   $0051
0356 90 02              BCC   NOCAR
0358 E6 56              INC   $0056
035A 46 52      NOCAR   LSR   $0052
035C 90 04              BCC   NOCRY
035E E6 56              INC   $0056
0360 E6 56              INC   $0056   DIVIDE X AND Y BY 2
0362 A9 01      NOCRY   LDAIM $01     DETERMINE QUADRANT OF NEW POINT
0364 A4 56      LOOP    LDY   $0056   AND PLACE QUADRANT NUMBER IN $0056
0366 F0 06              BEQ   MATCH
0368 0A                 ASLA
0369 C6 56              DEC   $0056
036B 4C 64 03           JMP   LOOP
```

```
036E 85 56     MATCH  STA   $0056
0370 06 52            ASL   $0052
0372 06 52            ASL   $0052
0374 06 52            ASL   $0052
0376 A5 52            LDA   $0052
0378 06 52            ASL   $0052
037A 26 53            ROL   $0053  MULTIPLY Y BY DECIMAL 40.
037C 06 52            ASL   $0052  (NO. CHARACTERS PER LINE)
037E 26 53            ROL   $0053
0380 65 52            ADC   $0052
0382 85 52            STA   $0052
0384 A5 53            LDA   $0053
0386 69 00            ADCIM $00
0388 85 53            STA   $0053
038A A5 52            LDA   $0052
038C 65 51            ADC   $0051  ADD X TO Y * 40.
038E 85 52            STA   $0052
0390 90 02            BCC   NOCHG
0392 E6 53            INC   $0053
0394 18        NOCHG  CLC
0395 A9 80            LDAIM $80
0397 65 53            ADC   $0053
0399 85 53            STA   $0053
039B A0 10            LDYIM $10    LOOK UP CHARACTER IN SCREEN
039D A2 00            LDXIM $00    POSITION X+Y*40 IN TABLE
039F A1 52            LDAIX $0052
03A1 88        CHARAC DEY
03A2 D9 BA 03         CMPY  TABLE
03A5 F0 09            BEQ   FOUND
03A7 C0 00            CPYIM $00
03A9 D0 F6            BNE   CHARAC
03AB A6 B1            LDX   $00B1  IF NOT IN TABLE, CHECK $B1 FOR
03AD F0 01            BEQ   FOUND  USR ARGUMENT
03AF 60               RTS

03B0 98        FOUND  TYA
03B1 05 56            ORA   $0056  COMPUTE NEW CHARACTER
03B3 A8               TAY
03B4 B9 BA 03         LDAY  TABLE  STORE NEW CHARACTER ON SCREEN
03B7 81 52            STAIX $C052
03B9 60               RTS

03BA 20        TABLE  =     $20    TABLE CONTAINS ALL SIXTEEN POSSIBLE
03BB 7E               =     $7E    PLOT CHARACTERS
03BC 7C               =     $7C
03BD E2               =     $E2
03BE 7B               =     $7B
03BF 61               =     $61
03C0 FF               =     $FF
03C1 EC               =     $EC
03C2 6C               =     $6C
03C3 7F               =     $7F
03C4 E1               =     $E1
03C5 FB               =     $FB
03C6 62               =     $62
03C7 FC               =     $FC
03C8 FE               =     $FE
03C9 A0               =     $A0
```

```
1   POKE  1,58:POKE 2,3
10  PRINT "(clr)"
20  FOR R=4 TO 16 STEP 4
30  P=38-R
40  Q=8+R
50  F=2*π/300
60  FOR I=0 TO 300
70  AN=F*I
80  X=INT(39.5+P*COS(AN))
90  Y=INT(24.5+Q*SIN(AN))
100 POKE 81,X:POKE 82,49-Y:A=USR(0)
110 NEXT I
120 NEXT R
130 GET G$:IF G$="" GOTO 130
```

FIGURE 1



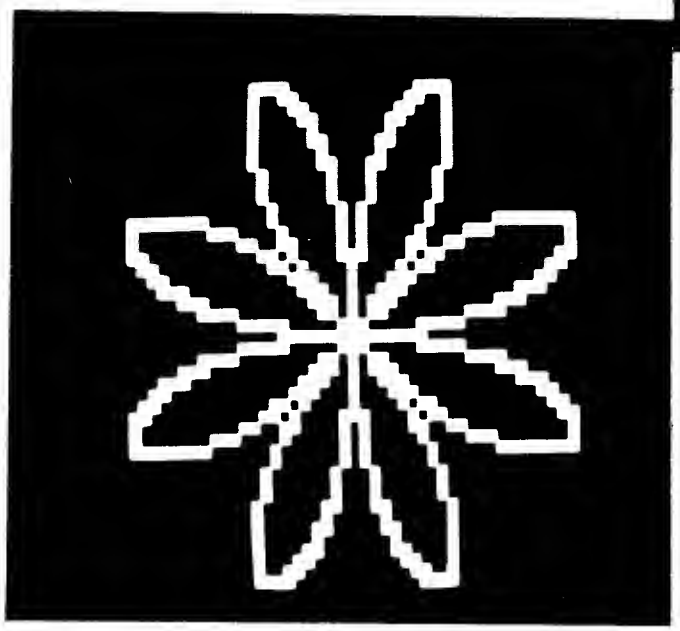


FIGURE 2A

```
1   POKE 1,58:POKE 2,3
10  PRINT "(clr)"
20  P=24:N=4
30  F=2*π/600
40  FOR I=0 TO 600
50  AN=I*F
55  R=P*SIN(N*AN)
60  X=INT(R*COS(AN)+39.5)
70  Y=INT(R*SIN(AN)+24.5)
80  POKE 81,X:POKE 82,49-Y:A=USR(0)
90  NEXT I
100 GET G$:IF G$="" GOTO 100
```

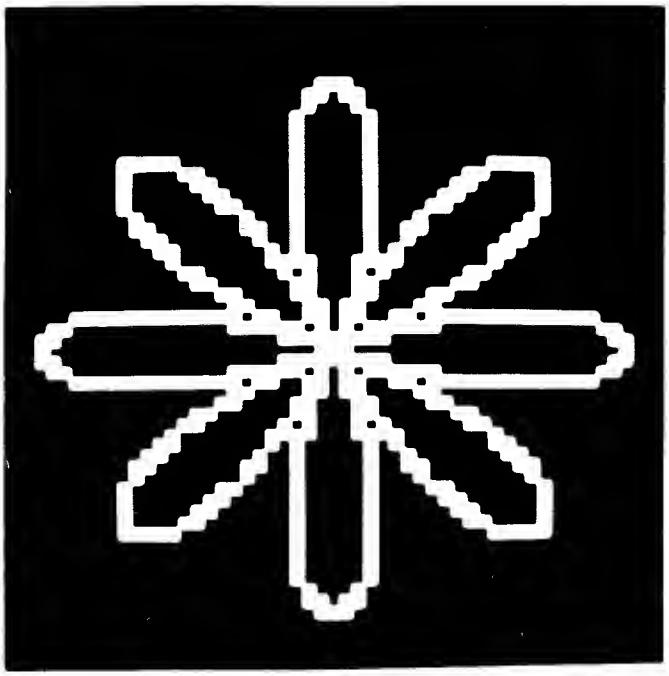


FIGURE 2B

(Changes to 2A only)

```
55 R=P*COS(N*AN)
```

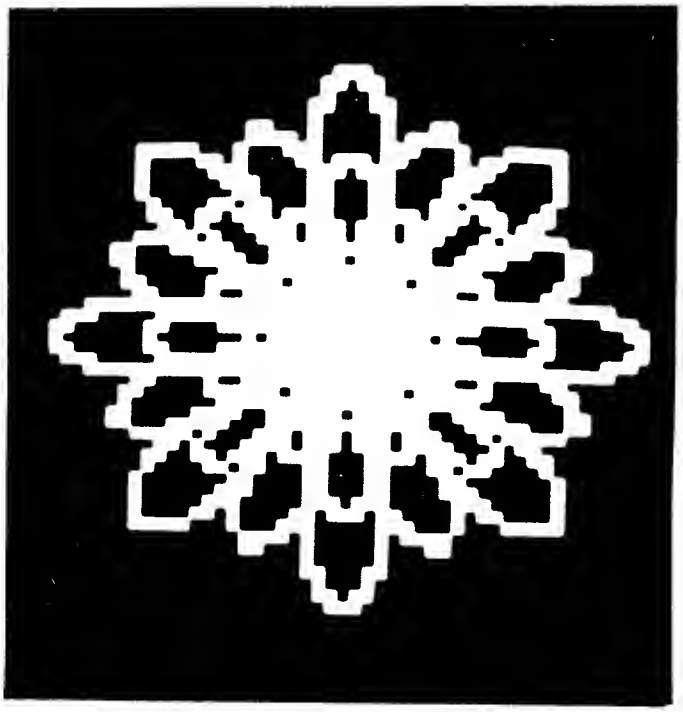


FIGURE 3

```
1   POKE 1,58:POKE 2,3
10  PRINT "(clr)"
20  P=9;Q=15/2
30  F=2*π/250
40  FOR I=0 TO 1250
50  AN=I*F
60  X=(P+Q)*COS(AN)+Q*COS((P+Q)*AN/Q)
70  Y=(P+Q)*SIN(AN)+Q*SIN((P+Q)*AN/Q)
80  X=INT(X+39.5):Y=INT(Y+24.5)
90  POKE 81,X:POKE 82,Y:A=USR(0)
100 NEXT I
110 GET G$: IF G$="" GOTO 110
```

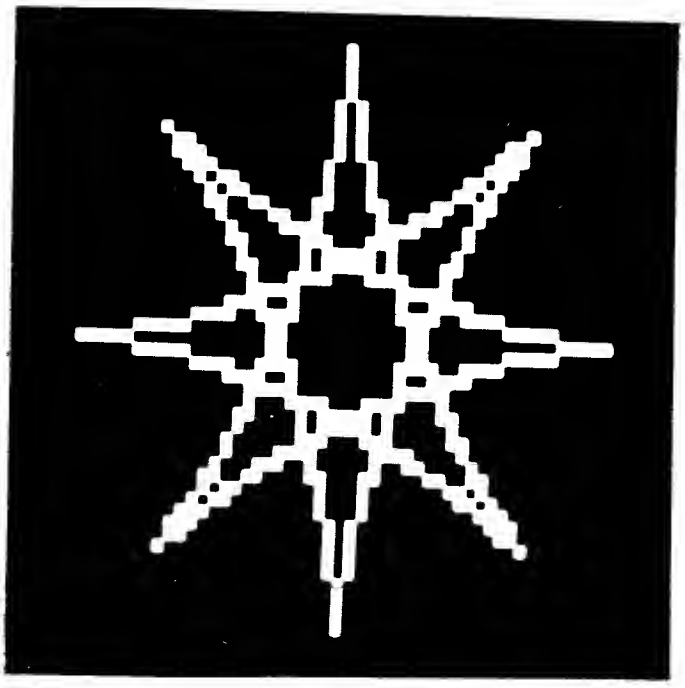
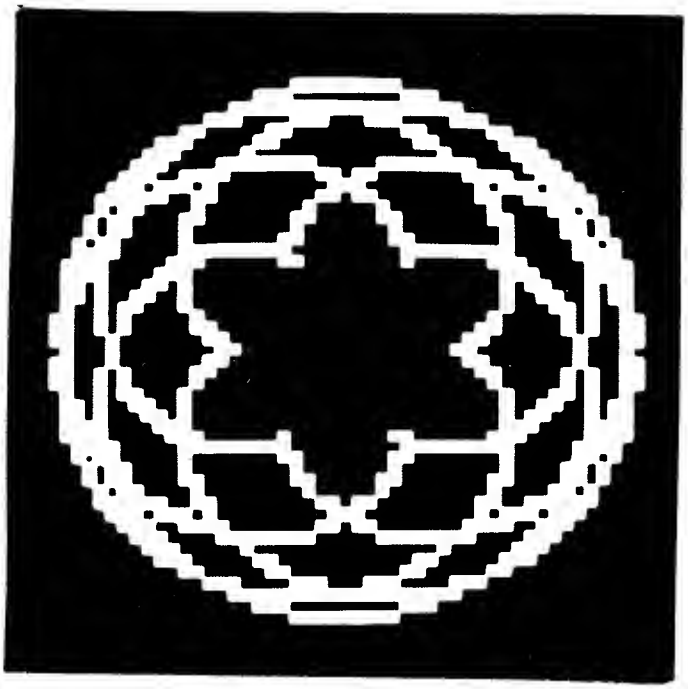


FIGURE 2C

(Changes to 2B only)

```
20  N=4
31  K1=1
32  FOR K2=0 TO 20 STEP 4
33  P=24-K2
34  K1=K1*-1
55  R=P*SIN(N*AN)
56  IF K1<0 THEN R=P*COS(N*AN)
```



```
1   POKE 1,58:POKE 2,3
10  PRINT "(clr)"
20  P=24:Q=9
22  DT=300
24  F=2*π/DT
28  FOR I=1 TO 25
30  DL=P*I/Q-INT(P*I/Q)
32  IF DL<.00001 GOTO 36
34  NEXT I
36  PT=I*P/Q
38  PRINT "(home)";INT(PT+.5)
40  FOR J=0 TO I*DT
50  AN=J*F
60  X=(P-Q)*COS(AN)+Q*COS((P-Q)*AN/Q)
70  Y=(P-Q)*SIN(AN-Q*SIN((P-Q)*AN/Q)
80  X=INT(X+39.5):Y=INT(Y+24.5)
90  POKE 81,X:POKE 82,Y:A=USR(0)
100 NEXT J
110 GET G$:IF G$="" GOTO 110
```

FIGURE 4

"THANKS FOR THE MEMORIES"
A PET MACHINE LANGUAGE MEMORY TEST

Harvey B. Herman
Chemistry Department
University of North Carolina at Greensboro
Greensboro, North Carolina 27412

Most people have surely heard the old Bob Hope theme song, "Thanks for the Memories." Whenever I hear it, I remind myself how much the explosion in personal computing is due to inexpensive memory chips. Several years ago I paid about $64.00 for a 4x16 (64 bits) static RAM by Intel. Today a 1x1024 static memory costs less than $2.00 - quite a hefty reduction in the per bit price.

That's the good news. The bad news is that all electronic parts occassionally fail and failures need to be diagnosed and repaired. The cheaper memory becomes the more we add and the harder and more time consuming it becomes to identify failed components. Diagnostic programs are one answer to this problem. Recently MICRO (7:25, Oct-Nov, 1978) published a PET memory test program written in BASIC. Execution time to test even about 200 bytes was quite long - about 1000 seconds. Clearly, a much faster test is necessary for even the smallest PET computers. If external memory is added the need for a much faster test becomes even more urgent.

An obvious way to increase the speed of a program is to write it in machine language. BASIC, a higher level of language, is notoriously slow especially when it must interpret each statement on every encounter. Writing faster machine language programs is facilitated with the help of a monitor program. PET owners have finally been given a free monitor program as part of their original purchase. This program has some nice features but the documentation is minimal. (How many times have we heard that song.) Important locations and subroutines are either not described at all or described sketchally so the program's usefulness to the average user is impaired.

However, not to worry. I have been experimenting with the monitor program by a combination of disassembly and trial and error have identified some of the missing links. You might guess from the title of this article that the purpose is to describe a fast machine language memory test. That is correct, but the other unspoken and possibly more important purpose is to teach the reader how better to use Commodore's machine language monitor program.

Table 1 summarizes important locations in Commodore's monitor. It is an expanded version of the table in their manual. For readers with access to the PET Gazette's LOMON program I have also included locations in that monitor which, incidentally, includes a disassembly in the latest version.

A large variety of machine language programs, including memory test programs, have appeared for other 6502 based systems. Jim Butterfield in "The First Book of KIM" (pp. 122-123) described a very fast machine language memory test program using a newly developed algorithm. I picked this particular KIM program for my first try at a PET translation program. Other programs developed for KIM (except when specifically hardware dependent) can be similarly translated. Our PETs will be more powerful than ever before as we can take already developed machine language software (the hard part), translate the programs for the PET and poke them into memory with the monitor (the easy part).

An inspection of the original KIM memory test program reveals some obvious PET incompatabilities. The KIM program originates at location zero and uses several KIM-specific locations (e.g., 1C4F as an exit to the KIM monitor). As a first pass we must relocate the program, change external jumps and substitute other page 0 locations. Table 2 shows the changes I made and gives some of my reasoning. Some decisions are self evident. For example, the second cassette buffer (starting at 033A) is a common place to store small PET programs as long as a second cassette is not being used. Other changes take advantage of specific features of the PET monitor. For example, the program counter (actually locations 22 and 23 as LOW and HI) is printed out after an exit to the monitor at location 0447. While the KIM monitor works similarly, the exit point and page zero locations printed are different and must be converted.

The translated program is executed using the CO command with a specified address (G 033A). After running the program several times, I became convinced it could be improved. Modifying a well documented program (as was the original) is, of course, much easier than writing one in the first place. The following changes were made:

1. Repeat the program continually until a key is pressed. Execution is very fast and one pass is not an adquate test.

2. Output an asterisk after each pass. It is nice to know the program is doing something.

3. Take the processor out of decimal mode, or hex arithmetic will not be done properly.

4. Input the beginning and ending page locations as a convenience in the GO step.

The last two modifications were easy to do before beginning execution. However, I occassionally forgot and felt it was better to insure it was done properly rather than to take a chance that the monitor had to be reloaded or BASIC restarted.

This version of the memory test program is also run with the GO command, with a specified address. The beginning and ending page location, separated by commas, are typed after the address (G 033A OA,1F). The program cycles until a faulty memory location is found which is printed as if it was the program counter or until any key is pressed. As advertised it is very fast a few second per pass for an 8 K PET (testing pages OA to 1F). A continuing outpouring of asterisks is very comforting.

My colleagues and I have found bad (or slow) memory chips with the original or modified test program on both KIM and PET computers. Happily, this does not happen very often; my hope is it won't happen to you. But if it does you will be prepared if you get this program running ahead of time. Good luck!

```
                    PET MEMORY TEST

                    BY HARVEY B. HERMAN
                    FEBRUARY 1979

                           ORG   $033A

                    BEGIN  *     $0023
                    END    *     $0024
                    POINTL *     $0019
                    POINTH *     $001A
                    FLAG   *     $001B
                    FLIP   *     $001C
                    MOD    *     $001D
                    PRINT  *     $FFD2
                    GET    *     $FFE4
                    INPUT  *     $FFCF
                    EXIT   *     $0447
                    ERROR  *     $049B
                    GTBYT  *     $0656

033A D8             START  CLD
033B 20 CF FF              JSR   INPUT
033E C9 20                 CMPIM $20    SPACE CHARACTER?
0340 F0 03                 BEQ   ABLE
0342 4C 9B 04              JMP   ERROR
0345 20 56 06       ABLE   JSR   GTBYT
0348 85 23                 STA   BEGIN
034A 20 CF FF              JSR   INPUT
034D C9 2C                 CMPIM $2C    COMMA ?
034F F0 03                 BEQ   BAKER
0351 4C 9B 04              JMP   ERROR
0354 20 56 06       BAKER  JSR   GTBYT
0357 85 24                 STA   END

0359 A9 00          LOOP   LDAIM $00
035B A8                    TAY
035C 85 19                 STA   POINTL
035E 85 1B          BIGLP  STA   FLAG
0360 A2 02                 LDXIM $02
0362 86 1D                 STX   MOD
0364 A5 23          PASS   LDA   BEGIN
0366 85 1A                 STA   POINTH
0368 A6 24                 LDX   END
036A A5 1B                 LDA   FLAG
036C 49 FF                 EORIM $FF
036E 85 1C                 STA   FLIP
0370 91 19          CLEAR  STAIY POINTL
0372 C8                    INY
0373 D0 FB                 BNE   CLEAR
0375 E6 1A                 INC   POINTH
0377 E4 1A                 CPX   POINTH
0379 B0 F5                 BCS   CLEAR
037B A6 1D                 LDX   MOD
037D A5 23                 LDA   BEGIN
037F 85 1A                 STA   POINTH
```

```
0381 A5 1B      FILL   LDA   FLAG
0383 CA         TOP    DEX
0384 10 04             BPL   SKIP
0386 A2 02             LDXIM $02
0388 91 19             STAIY POINTL
038A C8         SKIP   INY
038B D0 F6             BNE   TOP
038D E6 1A             INC   POINTH
038F A5 24             LDA   END
0391 C5 1A             CMP   POINTH
0393 B0 EC             BCS   FILL
0395 A5 23             LDA   BEGIN
0397 85 1A             STA   POINTH
0399 A6 1D             LDX   MOD
039B A5 1C      POP    LDA   FLIP
039D CA                DEX
039E 10 04             BPL   SLIP
03A0 A2 02             LDXIM $02
03A2 A5 1B             LDA   FLAG
03A4 D1 19      SLIP   CMPIY POINTL
03A6 D0 24             BNE   OUT
03A8 C8                INY
03A9 D0 F0             BNE   POP
03AB E6 1A             INC   POINTH
03AD A5 24             LDA   END
03AF C5 1A             CMP   POINTH
03B1 B0 E8             BCS   POP
03B3 C6 1D             DEC   MOD
03B5 10 AD             BPL   PASS
03B7 A5 1B             LDA   FLAG
03B9 49 FF             EORIM $FF
03BB 30 A1             BMI   BIGLP
03BD 84 19             STY   POINTL
03BF A9 2A             LDAIM $2A     ASTERISK CHARACTER *
03C1 20 D2 FF          JSR   PRINT
03C4 20 E4 FF          JSR   GET
03C7 F0 90             BEQ   LOOP
03C9 4C 47 04          JMP   EXIT

03CC 84 19      OUT    STY   POINTL
03CE 4C 47 04          JMP   EXIT
```

Program Notes

CTBYT	Change to $0658 for LOMON
033A	Clear decimal mode to insure arithmetic correct
033E	Compare with space character
033B - 0358	Input from screen: space, byte (2 characters), comma and byte. Store byte in begin and end page locations.
0359 - 03BE	Memory test program proper. Original author: Jim Butterfield.
03BF - 03CB	Print *, check for key press: no - repeat test yes - exit to monitor and print register buffer
03CC - 03D0	Abnormal exit to monitor. Program counter has address of fault.

MONITOR LOCATIONS

Table 1

Start of monitor	040F	
Exit to monitor	0447	
Break vector LOW 021B Normally 27		
Break vector HI 021C Normally 04		
Machine register storage buffer:		
Program counter LOW	0019	
Program counter HI	001A	The registers are initialized to the
Status register	001B	value in these locations after the G
Accumulator	001C	command. After the break instruction
X-index register	001D	(and break vector set to 0427) these
Y-index register	001E	locations will contain the final
Stack pointer	001F	values of the registers.
Operating System calls:		
Output byte (from A)	FFD2	
Input byte (left in A)	FFCF	(loc 260: 0 keyboard, 1 screen)
Get byte	FFE4	(A-0 no key depressed otherwide A- character)

	COMMODORE	LOMON (PET Gazette)
Output CR	04F2	04F2
Output space	063A	063B
Output byte as 2 hex	0613	0613
Input byte as 2 hex	065E	0660
ASCII to hex (from A)	0685	0687
Output? and wait for new command	049B	049B
Input 2 bytes as 4 hex (LOW in loc. 11, HI in 12)	064F	0651

KIM-PET EQUIVALENCES FOR THE MEMORY TEST PROGRAM

Table 2

	KIM	PET	NOTES
BEGIN	0000	0023	first two unused zero
END	0001	0024	page locations
POINTL	00FA	0019	printed as PC location
POINTH	00FB	001A	on exit
FLAG	0070	001B	printed as SR on exit
FLIP	0071	001C	printed as A on exit
MOD	0072	001D	printed as X on exit
EXIT	1C4F	0447	exit to monitor-print registers
START	0002	033A	start of second cassette buffer-well protected if device not used.

LIFESAVER

by J. Stelly
10918 Dunvegan Way
Houston, TX 77013

Is LIFE passing you by; does it progress so quickly than there is little time to enjoy it? Well, fear not--the LIFESAVER is here. Though time marches on, now you are in control. If you got "LIFE For Your PET" from Dr. Frank H. Covitz (**The Best of Micro**, p.65), LIFE moves along at a pretty good clip. LIFESAVER is a BASIC program that complements and provides some enhancements to Dr. Covitz machine language routines.

LIFESAVER provides a convenient grid for setting up cellular patterns, permits saving and loading of patterns on the built in cassette unit, and gives complete control of the time interval between generations. You may even single step through the LIFE sequences.

Commodore is supposedly mailing all owners of early model PET units the TIM monitor on cassette, so I will assume its availability in this discussion. It ain't the best monitor in the world, but it does allow you to load machine language programs directly from the cassette without any special loader routines. This does not exclude other methods the reader may have at his (or her) disposal if TIM is not available.

A single modification to Dr. Covitz program is required before it can be used with LIFESAVER. Location 191D (16) should be changed to read:

191D 60 RTS

When this change is made the program may be entered at 190A(16) e.g. SYS(6410). If the TIM monitor is used, simply do a hex dump of the machine language listing and save the program on tape using the instructions given in the manual.

Before loading LIFE (Dr. Covitz program) or LIEESAVER (by yours truly) from cassette, I recommend the following command be executed:

POKE 134,0:POKE 135,24

This lowers the BASIC boundary and prevents conflicts between the two programs. The regular BASIC limit can later be reinstated by POKE 135,32. It is also a good idea to load LIFE before LIFESAVER is loaded. This prevents the data pointer from getting initialized to the wrong location.

It may be possible to eliminate lines 3015 and 3035 from the BASIC listing, if you have a relatively late model PET. These lines are necessary for the older units that have a problem with writing file headers and cassette motor start/stop control. My unit was delivered in Sept. '78 and I was able to eliminate these lines.

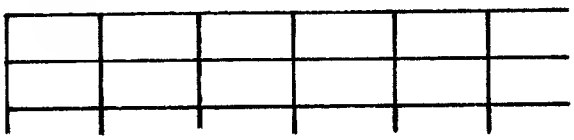
Assuming that both LIFE and LIFESAVER have successfully been loaded, you may begin entering your favorite cell patterns. Please refer to Dialog 1 (human inputs are underlined) to see how this is done. After the grid is printed simply press the 'RETURN' key and enter your pattern anywhere in the grid area using the cursor keys and the dot (•) symbol above the Q key. After you've created the desired pattern press the 'HOME' key and the 'RETURN' key in succession. This neat little trick returns control to the LIFESAVER routine without having to explicitly key in the command 'GOTO 1000'. After the PET has saved the pattern internally the user then has the options to save it on tape, have the computer generate LIFE patterns as described in Dr. Covitz article, or scrap it and input a new pattern.

The options are relisted after the execution of any LIFESAVER command. Examples on exercising the different options are given in the remaining dialogs.

LIFESAVER should relieve the user from the tedium of having to manually reenter a LIFE pattern every time it is desired to run it. It should also encourage the user to experiment with various LIFE forms, some of which are quite dazzling.

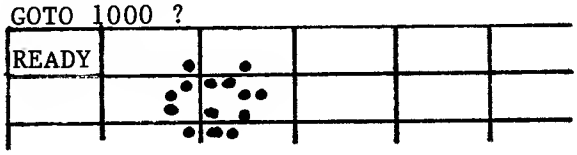
DIALOG 1

```
RUN

LIFE

PLEASE CHOOSE AN OPTION

1. CREATE A PATTERN
2. RUN LIFE GENERATOR
3. LOAD A PATTERN FROM CASSETTE
4. SAVE A PATTERN ON CASSETTE

OPTION NUMBER ? 1 (RETURN)

     (SCREEN CLEARS, THEN ...)

GOTO 1000 ?
```



(At this point the user hits the RETURN key and proceeds to input a cell pattern.)

GOTO 1000 ?



(With the desired pattern on the CRT the user presses the HOME and RETURN keys to resume program execution.)

STORING CELL PATTERN

(After a slight delay the computer again responds with the option list.)

DIALOG 2

```
LIFE
  .
  .
(Option List)
  .
  .
OPTION NUMBER ? 2 (RETURN)

(Screen clears ...)

HOW MANY GENERATIONS ? 7
DEVELOPMENT RATE
0:SINGLE STEP VIA (G) KEY
1-99:INTERMEDIATE RATES
100:MAX (255 GENERATION LIMIT)

RATE ? 75

(The computer proceeds to display
generations sequentially at the
specified rate.  The larger the
numerical value of the rate the
faster the generations are produced.
A rate of 0 meanc that only one
generation is produced at a time.
The G key must be pressed to obtain
subsequent generations.)
```

DIALOG 3

```
LIFE
  .
  .
(Option list)
  .
  .
OPTION NUMBER ? 4

(Screen clears ...)

HOW MANY
PATTERN NAME ? CHESIRECAT (RETURN)

(Pattern is saved and the option
list is printed.)
```

NOTE: In the following BASIC listing the lower case abbreviations stand for cursor control keys and have the following meaning:

clr = clear screen
home = home up
cd = cursor down
s = space key

LISTING

```
1    REM LIFESAVER
2    REM BY JAMES W. STELLY
3    REM POKE 135,24 BEFORE USING
100  DIM A$(25)
110  PRINT "clrLIFE":PRINT
120  PRINT "PLEASE CHOOSE AN OPTION:":PRINT
130  PRINT "1. CREATE A PATTERN"
140  PRINT "2. RUN LIFE GENERATOR"
150  PRINT "3. LOAD A PATTERN FROM CASSETTE"
160  PRINT "4. SAVE PATTERN ON CASSETTE"
170  INPUT "OPTION NUMBER";N
180  ON N GOSUB 200,2000,4000,3000
190  GOTO 110

     CREATE GRID FOR PATTERN INPUT

200  PRINT "clr cd";
210  FOR I=1 TO 5
220  PRINT "┌───┬───┬───┬───┬───┬───┬───┬───┬───┬───┐
230  PRINT "│   │   │   │   │   │   │   │   │   │   │
240  PRINT "│   │   │   │   │   │   │   │   │   │   │
250  PRINT ":
260  NEXT I
270  PRINT "├───┼───┼───┼───┼───┼───┼───┼───┼───┼───┤
280  PRINT "│   │   │   │   │   │   │   │   │   │   │
290  PRINT "│   │   │   │   │   │   │   │   │   │   │
300  PRINT "└───┴───┴───┴───┴───┴───┴───┴───┴───┴───┘
310  INPUT "homeGOTO 1000";A$
```

```
     STORE PATTERN
1000 PRINT "homeSTORING CELL PATTERN"
1010 FOR I=1 TO 24:A$(I)="":NEXT I
1020 FOR I=1 TO 24:FOR J=1 TO 39
1030 IF PEEK(32767+J+(I*40))= 81 THEN A$(I)=A$(I)+"●":GOTO 1050
1040 A$(I)=A$(I)+"-"
1050 NEXT J:NEXT I
1060 RETURN

     ACCESS LIFE GENERATOR
2000 INPUT "clrHOW MANY GENERATIONS";G
2010 PRINT "cdDEVELOPEMENT RATE:":PRINT
2020 PRINT "0;SINGLE STEP VIA (G) KEY"
2030 PRINT "1-99:INTERMEDIATE RATES"
2040 PRINT "100:MAX (255 GENERATIONS LIMIT)"
2050 INPUT "cdRATE";S
2060 PRINT "clrGEN 0"
2070 FOR I=1 TO 23:PRINT A$(I): NEXT I
2075 PRINT A$(I);:FOR I=1 TO 2000:NEXT I
2080 IF S=100 THEN POKE 6483,256-G:SYS(6410):GOTO 2140

     INTERMEDIATE RATES
2100 POKE 6483,255:IF S=0 GOTO 2160
2110 S=100-S:FOR I=1 TO G
2120 SYS(6410):PRINT "homeGEN";I
2130 FOR J=1 TO S*30:NEXT J:NEXT I
2140 GET A$:IF A$<>"X" GOTO 2140
2150 RETURN

     SINGLE STEP
2160 G=1
2170 SYS(6410):PRINT "homeGEN";G
2180 GET A$: IF A$="X" THEN RETURN
2190 IF A$="G" THEN G=G+1: GOTO 2170
220  GOTO 2180

     SAVE PATTERN
3000 INPUT "clrPATTERN NAME";A$
3010 OPEN 1,1,1,A$
3015 POKE 243,122:POKE 244,2
3020 FOR I=1 TO 24
3030 PRINT#1,A$(I)
3035 POKE 59411,53
3040 NEXT I
3050 CLOSE 1:RETURN

     LOAD PATTERN
4000 INPUT "clrPATTERN NAME";A$
4010 OPEN 1,1,0,A$
4020 FOR I=1 TO 24:INPUT#1,A$(I):NEXT I
4030 CLOSE 1:RETURN
```

THE ULTIMATE PET RENUMBER

Don Rindsberg
The Bit Stop
Box 973
Mobile, Alabama 36601

This article can be of help to the BASIC programmer in providing a fast, fool-proof renumbering system, but it also includes details on how to use the PET BASIC interpreter's own machine-language routines to do some useful chores.

Renumbering programs written in BASIC, such as Jim Butterfield's (see MICRO Dec 78 - Jan 79) are very slow in renumbering long programs, and because BASIC is cumbersome in performing such routine chores, the machine-language approach has some major advantages. This routine will renumber a 300-line program in around 20 seconds, as compared to more than 300 seconds for Jim's BASIC version. Further, Jim is forced to duck the issue of providing space for extra-digit line numbers, whereas by calling BASIC's line insertion routine, this program provides enough space for five digits for every GOTO, GOSUB, etc.

The entire program for renumbering is given in hexadecimal in listing 1. More later about how to enter it into your machine. With your BASIC program and the renumber routine in RAM, press SYS8181 (by coincidence, the name of the program) and you will either get a message of reassurance that all has gone well, or will get an error message, such as "line too long". In no case will the program bomb, because this is a two-pass program; during the first pass, nothing is done to the Basic text, other than making sure there is enough space for five-digit line numbers. If any problem exists, the BASIC text is unchanged.

DEVELOPING THE PROGRAM

Commodore made it a formidable task to decipher the code of BASIC sufficiently to be able to make patches for a short renumber system. The first obstacle is that the PEEK statement is disabled for the area of memory where BASIC resides. But, by sleight-of-hand, a little PUNCHing and POKFing and addition of a simple output port on PET's memory-expansion connector, the PET disgorged the contents of its ROMs into my homebrew machine and onto a disk; now, with the capability of having the programs in RAM, where breaks could be inserted for diagnosis, the job became a little easier.

Programming a renumber routine is made tedious by the fact that, in the BASIC text, the line numbers following the GOTO tokens are coded in ASCII, whereas the line numbers at the beginning of a line of text are coded as two-byte hex numbers. Fortunately, the BASIC interpreter has rountines built in to do these conversions back and forth between ASCII and hex. The locations of these and other routines called by this program are given in TABLE 1. Another problem encountered was locating some page zero registers, essential to 6502 programming, which are not altered by the BASIC itself. In some cases, I use space in the line buffer at 000A-0059, but this cannot be done in the section of the Program which uses the line buffer for its original purpose, i.e., inserting a line in its proper place in the BASIC text.

This program uses very little RAM, since no tables are created.

PROGRAM OPERATION

The program first sets or clears a flag, depending upon entry point (DCM 8181 or 8184), since entry point determines whether the renumber job is standard or custom. It then checks to see if sufficient memory exists to allow for insertion of spaces for as many as five digits for GOTO line numbers. An error message (see TABLE 2) is generated if there is less than one page available for this enlargement of the program. Then, each line of text is moved into the line buffer, and if a GOTO, GOSUB, or THEN (followed by a number) is present, spaces are inserted and the expanded line is inserted by BASIC's own line-insertion routine into the text area, just as though you retyped the line on the keyboard. Any lines too long for this expansion produce an error message before any harm is done to the program. BASIC's own error routine is called to print these messages! The "TOO LONG" message is a shortened version of "STRING TOO LONG" used by BASIC.

In the text, all statements are compressed into single-byte tokens, which I have listed for your reference in TABLE 3. For example, GOSUB is hex 8D, THEN is A7, etc. This program searches out all the 89, 8D and A7 tokens. Getting the proper ASCII numbers after these tokens requires conversion of the ASCII to hexadecimal and searching for a matching line number in the text area. If no match is found, the guy evidently had a GOTO pointing to a non-existent line number, so we flag this in the text by an opening parenthesis, such as:

```
GOTO(
GOSUB( :X=X+1
IF A=B THEN(
ON X GOTO 1234,(     ,5678,9987
```

When the progra is listed or run, the need for correction is obvious. While we are searching for a matching line number, we keep track of the new line number which corresponds to the current position in the text, so that when the match is found, the new line number can be converted to ASCII and placed directly into the text.

The actual resequencing process which follows is an anticlimax, because it requires so little coding (1E16-1E3E). When the entire renumbering job is done, we jump back to BASIC's warm start location.

USING THE PROGRAM

If you would like to renumber your program with the standard starting line number 100 and increment by 10, simply type SYS8181, which directs the program to hex address 1FF5. If you would like to choose a different starting line number or increment, POKE the desired values at the addresses shown in LISTING 2, and type SYS8184 to enter the program at 1FF8. If your BASIC program is long, it may take 3-4 seconds to complete the renumbering job. After renumbering, running the program will generally write over the renumber code, since it occupies the same space as some BASIC variables. The only precaution to be taken in renumbering is to avoid line numbers which exceed PET's limit of 63999.

ROUTINE ENTRY POINT (HFX)	FUNCTION AND IMPLEMENTATION
C359	Print an error message from the message table. Enter with X containing the location of the message relative to C190. Message terminator is ASCII having bit 7 on.
1F00	A duplicate of the original BASIC line insertion routine located at C3B4, except for the exit jump. Enter with the line assembled in the line buffer 000A-0059 with 00 as line terminator. Also, the character count must be in 005C, and the line number (hex) at 0008/9.
CCA4	Evaluate an expression whose beginning address is in 00C9/CA. We use this sub to convert from ASCII to binary, with the result appearing in the floating accumulator 00B0†.
DB1B	Convert fixed number in 00B1/2 to floating number. Enter with X=90 and carry set.
D6D0	Convert binary value, such as line number, in floating accumulator to two-byte fixed number and place in 0008/9.
DCAF	Convert floating number at 00B0† to ASCII and place in a string starting at 0101, preceded by a space or minus sign at 0100 and terminated by 00.
C38B	BASIC warm start. Prints READY.
CA27	Print message. Enter with ADH in Y, ADL in A. Message is ASCII string enough with 00.
DC9F	Print the decimal integer whose hex value is in microprocessor registers A and X, for example, a line number.

TABLE 1 - BASIC ROUTINES USED

MESSAGE	INTERPRETATION
CHECK FOR GOTO(ETC	Successful renumbering.
120 ? TOO LONG ERROR	Line 120 is too long to renumber. Break into two or more lines, and renumber again.
? OUT OF MEMORY ERROR	Program too long to renumber.
? SYNTAX ERROR	Attempt to RUN program with GOTO(remaining in program, or attempt to renumber with one of these in program text.
GOTO(GOSUB(ON X GOTO(IF A=B THEN(	The opening parenthesis in the text represents attempt to reference a non-existent line number.

Note: Lines of the following form are likely to cause a TOO LONG error:

100 ON X GOSUB 1,2,3,4,5,6,7,8,9,10,11,12

TABLE 2 - MESSAGES

```
                RENUMB ORG   $1D00
                   DON RINDSBERG
                   (C) 1978 N.A.I.L.

                      (& SIGN MEANS PLUS)

                 EXTERNAL ROUTINES

INSERT .      $1F00  INSERT A LINE INTO TEXT
MESSG  .      $1FCA  DONE MESSAGE

                 TEMPORARIES

BUFF   .      $0008  LINE BUFFER LOCATION
POINT  .      $0019  TEMP LINE BUFF POINTER
POINTX .      $001A  TEMP POINTER
LINCNT .      $005C  NO. CHAR. IN LINE
PTRSO  .      $007A  ORIGINAL POINTERS
PTRS   .      $006A  WORKING POINTERS
FLAG   .      $0069  FLAG THE GOTOS
BUFPTR .      $006E  LINE BUFF POINTER PAGE ZERO
COUNT  .      $006F  COUNTER
STARTC .      $00DB  CUSTOM STARTING LINE NO.
INTC   .      $00DD  CUSTOM INTERVAL
CUSTOM .      $00DE  FLAG CUSTOM JOB

                 BASIC PARAMETERS

FACC   .      $00B0  BASIC FLOATING ACCUM
BASICP .      $00C9  BASIC POINTER
BERROR .      $C359  BASIC ERROR ROUTINE
WARM   .      $C38B  BASIC WARM START
PRINT  .      $CA27  BASIC PRINT ROUTINE
EVAL   .      $CCA4  EXPRESSION EVALUATOR
FIX    .      $D6D0  CONVERT TO FIXED DP
FLOAT  .      $DB1B  CONVERT FIXED NMBR TO FLOAT
PNUMBR .      $DC9F  BASIC PRINT NUMBER
ASCII  .      $DCAF  CONVERT NMBR TO ASCII AT $0100

                 MAINLINE

1D00 A5 7D     START  LDA   PTRSO  &03 GET END TEXT ADH
1D02 C9 1B            CMPIM $1B   ENOUGH ROOM TO EXPAND?
1D04 90 05            BCC   SPACE
1D06 A2 52     BOMB   LDXIM $52   OUT OF MEMORY
1D08 4C FC 1E         JMP   ERROR
1D0B 20 BD 1E  SPACE  JSR   COPY   MAKE CC TEXT POINTERS
1D0E 20 3F 1E  NEXT   JSR   DNTST  ARE WE DONE THIS SECTION?
1D11 F0 2B            BEQ   RENUM
1D13 A2 08            LDXIM $08   LINE BUFFER START
```

```
1D15 A0 02           LDYIM $02    POINT TO LINE NMBR IN TEXT
1D17 B1 6A    GETBYT LDAIY PTRS   GET BYTE FROM TEXT
1D19 95 00           STAZX $00    STORE IN LINE BUFFER
1D1B C0 04           CPYIM $04    ZERO HERE NOT TERMINATOR
1D1D 90 04           BCC   SKIPA
1D1F C9 00           CMPIM $00
1D21 F0 04           BEQ   TERM   GOT THE TERMINATOR
1D23 C8       SKIPA  INY
1D24 E8              INX
1D25 D0 F0           BNE   GETBYT FORCED BRANCH
1D27 20 47 1E TERM   JSR   EDIT   EDIT ONE LINE
1D2A A5 69           LDAZ  FLAG
1D2C D0 0A           BNE   SKIPB  SKIP IF NO GOS FLAGGED
1D2E 38              SEC
1D2F A5 6E           LDA   BUFPTR
1D31 E9 05           SBCIM $05    CORRECT BYTE COUNT
1D33 85 5C           STA   LINCNT NEED CHAR COUNT
1D35 4C 00 1F        JMP   INSERT BUT RETURN TO NEXT LINE
1D38 20 C7 1E SKIPB  JSR   UPDATE POINT TO NEXT LINE
1D3B 4C 0E 1D        JMP   NEXT


1D3E 20 BD 1E RENUM  JSR   COPY   THE POINTERS
1D41 20 3F 1E NEXTR  JSR   DNTST  ARE WE DONE THIS PORTION?
1D44 D0 03           BNE   NOTDON
1D46 4C 16 1E        JMP   RESEQ
1D49 20 AE 1F NOTDON JSR   STRTLN GET STARTING LINE NMBR
1D4C A0 03    SCAN   LDYIM $03    POINT TO TEXT-1
1D4E C8       SCANA  INY
1D4F B1 6A    SCANX  LDAIY PTRS   GET A BYTE
1D51 D0 06           BNE   GOTEST BRANCH IF NOT TERMINATOR
1D53 20 C7 1E        JSR   UPDATE GO TO NEXT LINE
1D56 4C 41 1D        JMP   NEXTR
1D59 C9 89    GOTEST CMPIM $89    GOT A GOTO?
1D5B F0 15           BEQ   GOTO
1D5D C9 8D           CMPIM $8D    GOT A GOSUB?
1D5F F0 11           BEQ   GOTO
1D61 C9 A7           CMPIM $A7    GOT A THEN?
1D63 D0 E9           BNE   SCANA
1D65 C8       THEN   INY          POINT TO NEXT
1D66 B1 6A           LDAIY PTRS
1D68 C9 20           CMPIM $20    IGNORE SPACES
1D6A F0 F9           BEQ   THEN
1D6C 20 E5 1E        JSR   TSTDGT TEST FOR NUMBER
1D6F B0 E8           BCS   GOTEST
1D71 88              DEY
1D72 C8       GOTO   INY
1D73 84 19           STY   POINT  SAVE A MOMENT
1D75 98              TYA
1D76 18              CLC
1D77 65 6A           ADC   PTRS   POINT TO ASCII NMBRS
1D79 85 C9           STA   BASICP
1D7B 20 ED 1F        JSR   PATCH  BUG FIX
1D7E EA              NOP
1D7F 20 A4 CC        JSR   EVAL   CALL BASIC EVALUATOR
1D82 20 D0 D6        JSR   FIX    AND BASIC FIX ROUTINE
```

```
1D85 A5 7A      SEARCH LDA    PTRSO  SETUP SEARCH POINTERS
1D87 85 1A             STA    POINTX
1D89 A5 7B             LDA    PTRSO  &01
1D8B 85 1B             STA    POINTX &01
1D8D A0 00      SRCHLP LDYIM  $00
1D8F B1 1A             LDAIY  POINTX GET NEXT BYTE
1D91 C8                INY
1D92 11 1A             ORAIY  POINTX TEST FOR TWO ZERO BYTES
1D94 D0 10             BNE    NOTEND ZEROES MARK EOT
1D96 A9 20             LDAIM  $20    GET A SPACE
1D98 8D 00 01          STA    $0100  ASCII WORKSPACE
1D9B A9 28             LDAIM  $28    GET OPEN PAREN
1D9D 8D 01 01          STA    $0101
1DA0 88                DEY
1DA1 8C 02 01          STY    $0102  TERMINATE WITH ZERO
1DA4 F0 20             BEQ    MVASC  FORCED BRANCH
1DA6 A0 02      NOTEND LDYIM  $02
1DA8 B1 1A             LDAIY  POINTX GET LINE NO. LOW
1DAA C5 08             CMP    BUFF   MATCH?
1DAC D0 55             BNE    NOMAT
1DAE C8                INY
1DAF B1 1A             LDAIY  POINTX GET LINE NO. HIGH
1DB1 C5 09             CMP    BUFF   &01
1DB3 D0 4E             BNE    NOMAT
1DB5 A6 10      MATCH  LDX    BUFF   &08 GET CURRENT LINE NMBR
1DB7 86 B2             STX    FACC   &02
1DB9 A5 11             LDA    BUFF   &09 SECOND BYTE
1DBB 85 B1             STA    FACC   &01
1DBD A2 90             LDXIM  $90    SETUP FOR FLOAT
1DBF 38                SEC
1DC0 20 1B DB          JSR    FLOAT
1DC3 20 AF DC          JSR    ASCII  TO $0101 PLUS
1DC6 A2 FB      MVASC  LDXIM  $FB    MINUS 5
1DC8 A4 19             LDY    POINT
1DCA BD 06 00   LOOPA  LDAAX  $0006
1DCD F0 08             BEQ    BLANKS TERMINATOR ZERO
1DCF 91 6A             STAIY  PTRS
1DD1 C8                INY
1DD2 E8                INX
1DD3 D0 F5             BNE    LOOPA
1DD5 F0 0C             BEQ    COMMA
1DD7 A9 20      BLANKS LDAIM  $20    GET SPACE
1DD9 91 6A             STAIY  PTRS   STORE IT
1DDB C8                INY
1DDC E8                INX
1DDD D0 F8             BNE    BLANKS
1DDF 88                DEY
1DE0 D0 01             BNE    COMMA
1DE2 C8         COMMX  INY
1DE3 B1 6A      COMMA  LDAIY  PTRS   GET NEXT BYTE
1DE5 20 E5 1E          JSR    TSTDGT TEST FOR NUMBER
1DE8 B0 06             BCS    NOTNUM
1DEA A9 20             LDAIM  $20    SPACE
1DEC 91 6A             STAIY  PTRS   STORE IT
1DEE D0 F2             BNE    COMMX  FORCED
1DF0 C9 20      NOTNUM CMPIM  $20    SPACE?
```

```
1DF2 F0 EE            BEQ   COMMX
1DF4 C9 2C            CMPIM $2C     COMMA?
1DF6 08               PHP           DEFER TEST
1DF7 20 AE 1F         JSR   STRTLN GET STARTING LINE NMBR
1DFA 28               PLP           NOW TEST
1DFB D0 03            BNE   JSCANX NOT COMMA
1DFD 4C 72 1D         JMP   GOTO    GOT A COMMA
1E00 4C 4F 1D JSCANX  JMP   SCANX
1E03 20 EE 1E NOMAT   JSR   INCLIN INCR NEW LINE NMBR
1E06 A0 00            LDYIM $00
1E08 B1 1A            LDAIY POINTX GET NEXT LINE ADDRESS
1E0A 48               PHA
1E0B C8               INY
1E0C B1 1A            LDAIY POINTX
1E0E 85 1B            STA   POINTX &01
1E10 68               PLA
1E11 85 1A            STA   POINTX
1E13 4C 8D 1D         JMP   SRCHLP BACK TO SEARCH AGAIN


1E16 20 AE 1F RESEQ   JSR   STRTLN SETUP STARTING LINE
1E19 20 BD 1E         JSR   COPY    COPY THE POINTERS
1E1C 20 3F 1E LOOPR   JSR   DNTST   DONE?
1E1F F0 13            BEQ   WINDUP
1E21 A0 02            LDYIM $02     POINT TO LINE NMBR
1E23 A5 10            LDA   BUFF    &08 GET NEW ONE
1E25 91 6A            STAIY PTRS    STORE IT
1E27 C8               INY
1E28 A5 11            LDA   BUFF    &09
1E2A 91 6A            STAIY PTRS
1E2C 20 C7 1E         JSR   UPDATE ADVANCE TO NEXT LINE
1E2F 20 EE 1E         JSR   INCLIN INCREMENT LINE NMBR
1E32 90 E8            BCC   LOOPR   FORCED
1E34 A0 1F      WINDUP LDYIM MESSG  /100
1E36 A9 CA            LDAIM MESSG
1E38 20 27 CA         JSR   PRINT   END MESSAGE
1E3B 58               CLI           ALLOW KEYPRESSES
1E3C 4C 8B C3         JMP   WARM    BACK TO BASIC


1E3F A0 00      DNTST LDYIM $00
1E41 B1 6A            LDAIY PTRS    GET NEXT BYTE
1E43 C8               INY           ADVANCE TO NEXT
1E44 11 6A            ORAIY PTRS    OR WITH LAST TO FIND 0000
1E46 60               RTS


1E47 A2 09      EDIT  LDXIM BUFF    &01
1E49 86 6E            STX   BUFPTR
1E4B 86 69            STX   FLAG    SET FLAG
1E4D E6 6E      EDITX INC   BUFPTR
1E4F A6 6E            LDX   BUFPTR
1E51 B5 00            LDAZX $00
1E53 F0 71            BEQ   RTS
1E55 C9 89      EDITY CMPIM $89     GOTO?
1E57 F0 19            BEQ   SPACES
```

```
1E59 C9 8D          CMPIM $8D     GOSUB?
1E5B F0 15          BEQ   SPACES
1E5D C9 A7          CMPIM $A7     THEN?
1E5F D0 EC          BNE   EDITX   BACK FOR MORE
1E61 E6 6E    THENN INC   BUFPTR
1E63 A6 6E          LDX   BUFPTR
1E65 B5 00          LDAZX $00     BYTE AFTER THEN
1E67 C9 20          CMPIM $20     IGNORE SPACES
1E69 F0 F6          BEQ   THENN
1E6B 20 E5 1E       JSR   TSTDGT  IS IT NUMBER?
1E6E B0 E5          BCS   EDITY   IF NOT, GO BACK
1E70 C6 6E          DEC   BUFPTR
1E72 A2 09    SPACES LDXIM BUFF   &01 TEXT-1
1E74 E8       SPACEX INX
1E75 B5 00          LDAZX $00     LOOK FOR TERMINATOR
1E77 D0 FB          BNE   SPACEX
1E79 E0 54          CPXIM $54     LINE TOO LONG?
1E7B 90 0C          BCC   OKAY
1E7D A5 09          LDA   BUFF    &01
1E7F A6 08          LDX   BUFF    GET BAD LINE NMBR
1E81 20 9F DC       JSR   PNUMBR  PRINT IT
1E84 A2 BB          LDXIM $BB     TOO LONG MESSG
1E86 4C FC 1E       JMP   ERROR
1E89 A2 06    OKAY  LDXIM $06     DIGITS PLUS ONE
1E8B 86 6F          STX   COUNT
1E8D E6 6E    LOOP  INC   BUFPTR
1E8F C6 6F          DEC   COUNT
1E91 F0 12          BEQ   COMMAS
1E93 A6 6E          LDX   BUFPTR
1E95 B5 00          LDAZX $00
1E97 C9 20          CMPIM $20     TEST FOR SPACES
1E99 F0 F2          BEQ   LOOP
1E9B 20 E5 1E       JSR   TSTDGT  TEST FOR NUMBER
1E9E 90 ED          BCC   LOOP
1EA0 20 D5 1E       JSR   UPONE   MAKE ROOM FOR ONE DIGIT
1EA3 D0 E8          BNE   LOOP    FORCED BRANCH
1EA5 A0 00    COMMAS LDYIM $00
1EA7 84 69          STY   FLAG    WE WERE HERE
1EA9 A6 6E    FINDT LDX   BUFPTR
1EAB B5 00          LDAZX $00     FIND TERMINATOR
1EAD F0 17          BEQ   RTS
1EAF C9 20          CMPIM $20     SPACE?
1EB1 D0 04          BNE   TEST
1EB3 E6 6E          INC   BUFPTR
1EB5 D0 F2          BNE   FINDT   FORCED
1EB7 C9 2C    TEST  CMPIM $2C     COMMA?
1EB9 F0 B7          BEQ   SPACES
1EBB D0 90          BNE   EDITX


1EBD A2 04    COPY  LDXIM $04     COPY 4 BYTES
1EBF B5 79    LP    LDAZX $79
1EC1 95 69          STAZX $69     COPY POINTERS
1EC3 CA             DEX
1EC4 D0 F9          BNE   LP
1EC6 60       RTS   RTS
```

```
1EC7 A0 00      UPDATE LDYIM $00
1EC9 B1 6A             LDAIY PTRS    GET LINK ADL
1ECB 48                PHA           HOLD ON STACK
1ECC C8                INY
1ECD B1 6A             LDAIY PTRS    GET LINK ADH
1ECF 85 6B             STA   PTRS    &01 STORE LINK ADH
1ED1 68                PLA
1ED2 85 6A             STA   PTRS    STORE LINK ADL
1ED4 60                RTS


1ED5 A2 59      UPONE  LDXIM BUFF    &51 END BUFFER
1ED7 CA         LOOPU  DEX
1ED8 B5 00             LDAZX $00     GET A BYTE
1EDA 95 01             STAZX $01     MOVE UP ONE
1EDC E4 6E             CPX   BUFPTR
1EDE D0 F7             BNE   LOOPU
1EE0 A9 20             LDAIM $20     INSERT SPACE
1EE2 95 00             STAZX $00
1EE4 60                RTS


1EE5 C9 30      TSTDGT CMPIM '0
1EE7 90 03             BCC   SET
1EE9 C9 3A             CMPIM ':
1EEB 60                RTS           WITH CARRY CLEAR
1EEC 38         SET    SEC           CARRY SET IF NON-NMBR
1EED 60                RTS


1EEE 18         INCLIN CLC
1EEF A5 10             LDA   BUFF    &08
1EF1 65 12             ADC   BUFF    &0A
1EF3 85 10             STA   BUFF    &08
1EF5 A5 11             LDA   BUFF    &09
1EF7 69 00             ADCIM $00     ADD INTERVAL
1EF9 85 11             STA   BUFF    &09  TO CURRENT LINE
1EFB 60                RTS


1EFC 58         ERROR  CLI           ALLOW KEYPRESS
1EFD 4C 59 C3          JMP   BERROR  BASIC ERROR PROCESSOR


1FAE                   ORG   $1FAE


1FAE A9 64      STRTLN LDAIM $64     DEFAULT 100
1FB0 85 10             STA   BUFF    &08
1FB2 A9 00             LDAIM $00     HIGH ORDER
1FB4 85 11             STA   BUFF    &09
1FB6 A2 0A             LDXIM $0A     INTERVAL 10
1FB8 A5 DE             LDA   CUSTOM  TEST FOR CUSTOM
```

```
1FBA 10 0A            BPL   SKIPL
1FBC A6 DD            LDX   INTC    CUSTOM INTERVAL
1FBE A5 DB            LDA   STARTC CUSTOM START
1FC0 85 10            STA   BUFF    &08
1FC2 A5 DC            LDA   STARTC &01
1FC4 85 11            STA   BUFF    &09
1FC6 86 12      SKIPL STX   BUFF    &0A
1FC8 60               RTS
1FC9 EA               NOP

        FINAL MESSAGE $1FCA THROUGH $1FEC
        "CHECK FOR GOTO( ETC"

1FED            PATCH ORG   $1FED

1FED A5 6B            LDA   PTRS    &01
1FEF 69 00            ADCIM $00
1FF1 85 CA            STA   BASICP &01
1FF3 60               RTS
1FF4 EA               NOP

1FF5 18         ENTRY CLC           CLEAR FOR STANDARD
1FF6 90 01            BCC   ALL
1FF8 38         ENTRYA SEC          SET FOR CUSTOM
1FF9 78         ALL   SEI           DISABLE KEYS
1FFA 66 DE            RORZ  CUSTOM FLAG IN BIT 7
1FFC 4C 00 1D         JMP   START


       INSERT ORG   $1F00
         DUPLICATE  OF BASIC INSERT ROUTINE
         EXCEPT FOR EXIT JUMP


1F00 20 22 C5         JSR   $C522
1F03 90 44            BCC   INSC
1F05 A0 01            LDYIM $01
1F07 B1 AE            LDAIY $AE
1F09 85 72            STAZ  $72
1F0B A5 7C            LDAZ  $7C
1F0D 85 71            STAZ  $71
1F0F A5 AF            LDAZ  $AF
1F11 85 74            STAZ  $74
1F13 A5 AE            LDAZ  $AE
1F15 88               DEY
1F16 F1 AE            SBCIY $AE
1F18 18               CLC
1F19 65 7C            ADCZ  $7C
1F1B 85 7C            STAZ  $7C
1F1D 85 73            STAZ  $73
1F1F A5 7D            LDAZ  $7D
1F21 69 FF            ADCIM $FF
1F23 85 7D            STAZ  $7D
1F25 E5 AF            SBCZ  $AF
1F27 AA               TAX
1F28 38               SEC
1F29 A5 AE            LDAZ  $AE
1F2B E5 7C            SBCZ  $7C
1F2D A8               TAY
1F2E B0 03            BCS   INSA
1F30 E8               INX
1F31 C6 74            DECZ  $74
1F33 18         INSA  CLC
1F34 65 71            ADCZ  $71
1F36 90 03            BCC   INSB
1F38 C6 72            DECZ  $72
1F3A 18               CLC
1F3B B1 71      INSB  LDAIY $71
1F3D 91 73            STAIY $73
1F3F C8               INY
1F40 D0 F9            BNE   INSB
1F42 E6 72            INCZ  $72
1F44 E6 74            INCZ  $74
1F46 CA               DEX
1F47 D0 F2            BNE   INSB
1F49 A5 0A      INSC  LDAZ  $0A
1F4B F0 2F            BEQ   INSF
1F4D A5 86            LDAZ  $86
1F4F A4 87            LDYZ  $87
1F51 85 82            STAZ  $82
1F53 84 83            STYZ  $83
1F55 A5 7C            LDAZ  $7C
1F57 85 A9            STAZ  $A9
1F59 65 5C            ADCZ  $5C
1F5B 85 A7            STAZ  $A7
1F5D A4 7D            LDYZ  $7D
1F5F 84 AA            STYZ  $AA
1F61 90 01            BCC   INSD
1F63 C8               INY
1F64 84 A8      INSD  STYZ  $A8
1F66 20 DA C2         JSR   $C2DA
1F69 A5 80            LDAZ  $80
1F6B A4 81            LDYZ  $81
1F6D 85 7C            STAZ  $7C
1F6F 84 7D            STYZ  $7D
1F71 A4 5C            LDYZ  $5C
```

LOCATION HEX	DECIMAL	VALUE TO BE POKED
00DB	219	Low order starting line number (wieght 1)
00DC	220	High order starting line number (weight 256)
00DD	221	Increment desired (1-255)

Example: POKE 219,232
POKE 220,3
POKE 221,50

This will give a starting line number of 3 x 256 † 232 = 1000, and following lines will be incremented by 50.

LISTING 2 - NON-STANDARD LINE RENUMBER

STATEMENT	TOKEN	STATEMENT	TOKEN
END	80	FN	A5
FOR	81	SPC(	A6
NEXT	82	THEN	A7
DATA	83	NOT	A8
INPUT#	84	STEP	A9
INPUT	85	+	AA
DIM	86	-	AB
READ	87	*	AC
LET	88	/	AD
GOTO	89	↑	AE
RUN	8A	AND	AF
IF	8B	OR	B0
RESTORE	8C	>	B1
GOSUB	8D	=	B2
RETURN	8E	<	B3
REM	8F	SGN	B4
STOP	90	INT	B5
ON	91	ABS	B6
WAIT	92	USR	B7
LOAD	93	FRE	B8
SAVE	94	POS	B9
VERIFY	95	SQR	BA
DEF	96	RND	BB
POKE	97	LOG	BC
PRINT#	98	EXP	BD
PRINT	99	COS	BE
CONT	9A	SIN	BF
LIST	9B	TAN	C0
CLR	9C	ATN	C1
CMD	9D	PEEK	C2
SYS	9E	LEN	C3
OPEN	9F	STR$	C4
CLOSE	A0	VAL	C5
GET	A1	ASC	C6
NEW	A2	CHR$	C7
TAB(	A3	LEFT$	C8
TO	A4	RIGHT$	C9
		MID$	CA

TABLE 3

TOKENS (shorthand used in BASIC text)

ENTERING THE PROGRAM

The hard way to load the program into your PET is to convert my nex listing into decimal and POKE each byte into memory. This is, of course, a challenge to your accuracy and diligence, although it may take only slightly longer than renumbering by hand. It is only a little easier to write a BASIC program which will accept the hex data and convert to decimal, with the hex incorporated in DATA statements and obtained by the READ statement. With this alternate, the program can be recorded for future use.

To make loading painless (excpet for the wallet), I have arranged to make tapes available through NAIL*, Drawer F, Mobile, Alabama 36601. These tapes load the machine-language program directly into high memory. Ask for "SYS8181" and send $18.18. By the way, they also have a dandy PET monitor called SYS7171 for $29.71, which has machine language capabilities, the ability to co-reside in RAM with BASIC programs, but also has the very helpful feature of being able to APPEND one BASIC program to another, just like the big boys do, with interleaving of lines. Like SYS8181, it uses the BASIC line-inserting routine to do the merging, just as though you typed all those new lines on your keyboard. I used a version of this monitor to develop SYS8181. If there is sufficient interest out there, I may develop a ROM version of SYS8181, but you will have to be a hardware buff to wire it into your PET.

Since PET BASIC was written by the same company who write APPLESOFT and is similar, some APPLE owners may wish to obtain a disassembled, documented listing of this renumbering program from me for $5.00.

*National Artficial Intelligence Laboratory

```
1F73 88                 DEY
1F74 B9 06 00   INSE    LDAAY $0006
1F77 91 AE              STAIY $AE
1F79 88                 DEY
1F7A 10 F8              BPL   INSE
1F7C 20 67 C5   INSF    JSR   $C567
1F7F A5 7A              LDAZ  $7A
1F81 A4 7B              LDYZ  $7B
1F83 85 71              STAZ  $71
1F85 84 72              STYZ  $72
1F87 18                 CLC
1F88 A0 01      INSG    LDYIM $01
1F8A B1 71              LDAIY $71
1F8C D0 03              BNE   INSH
1F8E 4C 38 1D           JMP   $1D38
1F91 A0 04      INSH    LDYIM $04
1F93 C8         INSI    INY
1F94 B1 71              LDAIY $71
1F96 D0 FB              BNE   INSI
1F98 C8                 INY
1F99 98                 TYA
1F9A 65 71              ADCZ  $71
1F9C AA                 TAX
1F9D A0 00              LDYIM $00
1F9F 91 71              STAIY $71
1FA1 A5 72              LDAZ  $72
1FA3 69 00              ADCIM $00
1FA5 C8                 INY
1FA6 91 71              STAIY $71
1FA8 86 71              STXZ  $71
1FAA 85 72              STAZ  $72
1FAC 90 DA              BCC   INSG
```

A PET HEX DUMP PROGRAM

Joseph Donato
193 Walford Rd. E.
Sudbury, ONT., Canada

Have you PET owners ever wondered how it could be possible to look at your BASIC which resides in Read Only Memory (ROM)? To be able to look for routines entry points and other interesting codes in machine language?

This program will do just that. You can look at all memory locations in PET's BASIC which starts at 49152 decimal or COOO hexadecimal in memory. One is able for example to look at locations D71E through D890 where addition and subtraction routines are carried out, D8BF through D8FC where the log function is evaluated, D9E1 through DA73 where division is performed and many other locations where other routines are carried out.
A start for this program was provided by Mr. Herman's article of MICRO 7:47. Of course the same information was available in the Commodore Users Notes.

In any event I decided that the ultimate goal of the program would be to provide a memory dump of some sort in hexadecimal notation so that machine language instructions could easily be recognized.

The output of the program is formatted as a starting address followed by either 32 or 8 bytes of data per line, all in hexadecimal, depending on whether or not a printer is to be used. With the data bytes in hex notation it is very easy to correlate them with the 6502 microprocessor machine language instruction set.

The program listing has been thoroughly debugged and tested. Although the program was originally written for a PET with a Centronics printer, as I outlined in the REM's, the program will run on a "bare" PET with no problem.

The changes for a "bare" PET are as follows:

1. Omit line 10.
2. Change line 542 to read:
   542 IF L<>9 THEN 570
3. Omit all print statements and substitute instead the print format outlined in the REM's at lines 606 through 612. These print lines are to be placed at line 545, 546, 547, 548.
4. Notice that there is no comma or semicolon after the last print character. This is very important otherwise the format will be destroyed.

A considerable amount of time was spent on both versions of the program. No problems were encountered in running either version.

I hope that by following the machine language coding of the 6502 some of you will obtain a better understanding of PET's Basic 'inner workings'. Also some of you who have the T.I.M. monitor will be able to trace its subroutines and jumps to Basic. Perhaps it may inspire you in writing some machine language programs or routines.

I should add that if one wishes to look at different addresses other than the COOO (49152 decimal), all you need do is to change the starting address value "K" in line 240. This must be in decimal notation.

I hope you get as much pleasure as I did 'sneaking a look' at PET's Basic.

```
 1 REM *** A BASIC PET HEX DUMP ***
 2 REM THIS PROGRAM WILL PEEK AT PET'S
 3 REM MEMORY IN ROM STARTING AT A GIVEN ADDRESS 'K' (49152 DECIMAL) AND RETURN
 4 REM THE CORRESPONDING DATA.  ALL VALUES ARE CONVERTED TO HEXADECIMAL PRIOR TO
 5 REM PRINTING.  THE FORMAT IS: STARTING ADDRESS PLUS 32 OR 8 BYTES OF DATA,
 6 REM PER LINE DEPENDING WHETHER OR NOT A PRINTER IS USED.
 7 REM
 8 REM THE COMMAND ON LINE 10 INITIALIZES THE PRINTER PORT.  IT *MUST* BE OMITTED
 9 REM IF A "BARE" PET IS USED.
10 OPEN 5,5:CMD 5
11 REM FOLLOWING IS A MACHINE LANGUAGE
12 REM ROUTINE WHICH RESIDES IN NUMBER 2 TAPE
13 REM BUFFER AREA.  IT RETURNS THE CONTENTS OF THE CORRESPONDING MEMORY
14 REM LOCATIONS SPECIFIED BY 'K'.
15 POKE(1),58
16 POKE(2),3
17 POKE(826),32
20 POKE(827),167
30 POKE(828),208
40 POKE(829),166
```

```
50 POKE(830),179
60 POKE(831),164
70 POKE(832),180
80 POKE(833),134
90 POKE(834),180
100 POKE(835),132
120 POKE(836),179
130 POKE(837),162
140 POKE(838),00
150 POKE(839),161
160 POKE(840),179
170 POKE(841),168
180 POKE(842),169
190 POKE(843),00
200 POKE(844),32
210 POKE(845),120
220 POKE(846),210
230 POKE(847),96
232 REM SET UP STORAGE AREA FOR ONE
233 REM LINE OF HEX VALUES TO BE PRINTED
235 DIM N1$(40),NO$(40)
236 REM INITIALIZE CHARACTER COUNTER
237 L=1
238 REM THE VALUE OF 'K' DETERMINES
239 REM THE STARTING ADDRESS.
240 FOR K=49152 TO 65536
241 I=K
250 A=USR(K-65536)
255 REM LINES 270-530 CONSIST OF A SUBROUTINE TO CONVERT ALL VALUES FROM
256 REM DECIMAL TO HEXADECIMAL NOTATION
270 B%=16
280 D=A
390 H$="0123456789ABCDEF"
400 NO$(L)=""
405 N1$(L)=""
410 F%=LOG(I)/LOG(B%)
411 REM BECAUSE THE DECIMAL TO HEX ROUTINE
412 REM RETURNS A SINGLE '0' FOR VALUES
413 REM OF A=0, LINE 416 CONVERTS
414 REM ANY OF THESE ZERO VALUES TO
415 REM A DOUBLE HEX '00'.
416 IF A=0 THEN NO$(L)="00":GOTO 480
418 G%=LOG(D)/LOG(B%)
420 FOR J=G% TO 0 STEP -1
430 X=INT(B%^J)
440 C%=D/X
445 REM LINE 455 INSERTS A LEADING ZERO
446 REM IN HEXADECIMAL VALUES OF LESS
447 REM THAN 'F'(15). EX. '7'='07' ETC.
450 NO$(L)=NO$(L)+MID$(H$,C%+1,1)
455 IF A<16 THEN NO$(L)=('0'+NO$(L))
460 D=INT(D-C%*X)
470 NEXT J
480 FOR J=F% TO 0 STEP -1
490 X=INT(B%^J)
500 C%=INT(I/X)
510 N1$(L)=N1$(L)+MID$(H$,C%+1,1)
520 I=INT(I-C%*X)
530 NEXT J
```

```
532 REM SUBROUTINE FOR DECIMAL TO HEXADECIMAL CONVERSION ENDS HERE
535 L=L+1
536 REM LINE 542 CHECKS TO SEE IF THE
537 REM REQUIRED NUMBER OF CHARACTERS
538 PER LINE HAVE BEEN DONE.  THE TEST VALUE
539 NUMBER 33 *MUST* BE CHANGED TO A NUMBER 9 IF A "BARE" PET IS USED.
542 IF L<>33 THEN 570
545 PRINT N1$(1),"  ",NO$(1),"  ",NO$(2),"  ",NO$(3),"  ",NO$(4),"  ",NO$(5),
546 PRINT "  ",NO$(6),"  ",NO$(7),"  ",NO$(8),"  ",NO$(9),"  ",NO$(10),"  ",
547 PRINT NO$(11),"  ",NO$(12),"  ",NO$(13),"  ",NO$(14),"  ",NO$(15),"  ",
548 PRINT NO$(16),"  ",NO$(17),"  ",NO$(18),"  ",NO$(19),"  ",NO$(20),"  ",
549 PRINT NO$(21),"  ",NO$(22),"  ",NO$(23),"  ",NO$(24),"  ",NO$(25),"  ",
550 PRINT NO$(26),"  ",NO$(27),"  ",NO$(28),"  ",NO$(29),"  ",NO$(30),"  ",
560 PRINT NO$(31),"  ",NO$(32)
565 L=1
570 NEXT K
600 REM THE PRINT STATEMENT FOR THE PET
602 REM WITH NO PRINTER "BARE" SHOULD BE AS FOLLOWS:
606 REM PRINT N1$(1);"  ";NO$(1),"  ";
608 REM NO$(2);"  ";NO$(3);"  "NO$(4);
610 REM "  ";NO$(5);"  ";NO$(6);"  ";
612 REM NO$(7);"  ";NO$(8);"  ";NO$(9)
615 END
```

```
C000 1D C7 48 C6 35 CC EF C7 C5 CA DF CA 70 CF 23 CB 9C C8 9C C7 74 C7 1F C8 AC C7 77 C7 C9 C7 32 C8
C020 1B C7 42 C8 01 D7 D4 FF D7 FF DA FF 94 D2 F8 D6 7E C9 9E C9 44 C7 A7 C5 6F C7 84 C9 DD FF BF FF
C040 C2 FF 9E CA 50 C5 9C DB 9E D8 2A D8 00 00 64 D2 85 D2 24 DE 45 DF BF D8 A0 DE 9E DF A5 DF EE DF
C060 48 E0 E6 D6 54 D6 49 D3 85 D6 63 D6 C4 D5 D8 D5 04 D6 0F D6 79 3E D7 73 27 D7 7B FF D8 7B E3 D9
C080 7F 2D DE 50 D8 CE 46 D5 CE 7D 66 DE 5A E7 CD 64 85 CF 45 4E C4 46 4F D2 4E 45 58 D4 44 41 54 C1
C0A0 49 4E 50 55 54 A3 49 4E 50 55 D4 44 49 CD 52 45 41 C4 4C 45 C4 47 4F 54 CF 52 55 CE 49 C6 52 45
C0C0 53 54 4F 52 C5 47 4F 53 55 C2 52 45 54 55 52 CE 52 45 CD 53 54 4F D0 4F CE 57 41 49 D4 4C 4F 41
C0E0 C4 53 41 56 C5 56 45 52 49 46 D9 44 45 C6 50 4F 4B C5 50 52 49 4E 54 A3 50 52 49 4E D4 43 4F 4E
C100 D4 4C 49 53 D4 43 4C D2 43 4D C4 53 59 D3 4F 50 45 CE 43 4C 4F 53 C5 47 45 D4 4E 45 D7 54 41 42
C120 A8 54 CF 46 CE 3C 50 43 A8 54 48 45 CE 4E 4F D4 53 54 45 D0 AB A0 AA AF DE 41 4E C4 4F D2 BE B0
C140 BC 53 47 CE 49 4E D4 41 42 D3 55 53 D2 46 52 C5 59 4F D3 53 51 D2 52 4E C4 4C 4F C7 45 58 D0 43
C160 4F D3 53 49 CE 54 41 CE 41 54 CE 50 45 45 CB 4C 45 CE 53 54 52 A4 56 41 CC 41 3C C3 43 48 52 A4
C180 4C 45 46 54 A4 52 49 47 48 54 A4 40 49 44 A4 00 4E 45 58 54 20 57 49 54 48 4F 55 54 20 46 4F D2
C1A0 53 59 4E 54 41 D8 52 45 54 55 52 4E 20 57 49 54 48 4F 55 54 20 47 4F 53 55 C2 4F 55 54 20 4F 46
C1C0 20 44 41 54 A1 49 4C 4C 45 47 41 4C 20 51 55 41 4E 54 49 54 D9 00 00 00 00 00 4F 56 45 52 46 4C
C1E0 4F D7 4F 55 54 20 4F 46 20 4D 45 4D 4F 52 D9 55 4E 44 45 46 27 44 20 53 54 41 54 45 4D 45 4E D4
C200 42 41 44 20 53 55 42 53 43 52 49 50 D4 52 45 44 49 4D 27 44 28 41 52 52 41 D9 44 49 56 49 53 49
C220 4F 4E 20 42 59 20 5A 45 52 CF 49 4C 4C 45 47 41 4C 20 44 49 52 45 43 D4 54 59 50 45 20 4D 49 53
C240 4D 41 54 43 C8 53 54 52 49 4E 47 20 54 4F 4F 20 4C 4F 4E C7 42 41 44 20 44 41 54 A1 46 4F 52 4D
C260 55 4C 41 28 54 4F 4F 20 43 4F 4D 50 4C 45 D8 43 41 4E 27 54 20 43 4F 4E 54 49 4E 55 C5 55 4E 44
C280 45 46 27 44 20 46 55 4E 43 54 49 4F CE 20 45 52 52 4F 52 00 20 49 4E 20 00 0D 0A 52 45 41 44 59
C2A0 2E 00 0A 00 0D 0A 42 52 45 41 4B 00 BA E8 E8 E8 E8 BD 01 01 C9 81 D0 21 A5 99 D0 0A BD 02 01 85
C2C0 98 BD 02 01 85 99 DD 03 01 D0 07 A5 98 D0 02 01 F0 07 8A 18 69 12 AA D0 D8 60 20 2A C3 85 80 84
C2E0 81 38 A5 A9 E5 AE 85 71 A8 A5 AA E5 AF AA E8 98 F0 23 A5 A9 38 E5 71 85 A9 B0 03 C6 AA 38 A5 A7
C300 E5 71 85 A7 B0 08 C6 A8 90 04 B1 A9 91 A7 88 D0 F9 B1 A9 91 A7 C6 AA C6 A8 CA D0 F2 60 0A 69 36
C320 B0 35 85 71 8A E4 71 90 2E 60 C4 83 90 28 D0 04 C5 82 90 22 48 A2 09 98 48 B5 A6 CA 10 FA 20 04
C340 D4 A2 F7 68 95 B0 E8 30 FA 68 A8 68 C4 83 90 06 D0 05 C5 82 B0 01 60 A2 52 46 64 A5 03 F0 07 20
C360 CC FF A9 00 85 03 20 D2 C9 20 47 CA BD 90 C1 48 29 7F 20 49 CA E8 68 10 F3 20 94 C5 A9 80 A0 C2
C380 20 27 CA A4 89 C8 F0 03 20 94 DC 46 64 A9 99 A0 C2 20 27 CA 20 68 C4 86 C9 84 CA 20 C2 00 F0 F4
C3A0 A2 FF 86 89 90 06 20 80 C4 4C E9 C6 20 63 C8 20 80 C4 84 5C 20 22 C5 90 44 A6 01 B1 AE 65 72 A5

BREAK IN 240
READY.
```

Example of a partial Hex Dump obtained with the Program

CONTINUOUS MOTION GRAPHICS OR HOW TO FAKE A JOYSTICK WITH THE PET

Alan K. Christensen
1303 Suffolk
Austin, TX 78723

When using the PET graphics to represent motion it becomes apparent that the BASIC supported routines are not fast enough to allow smooth movement. If the keyboard and screen are accessed directly the appearance of controlled motion can be greatly enhanced. As an example I will use a short game written in BASIC although the techniques can be used by machine language programs with even better results.

Let me first describe the game and then explain how the effects are produced. The initial appearance of the screen is two walls at the right and left sides of the screen with a ball and pound sign (#) which I will refer to as a bat (see figure 1). The ball goes into motion and appears to bounce off the top and bottom of the screen and the walls. Each time the ball strikes a wall it causes part of the wall to disappear. The ball will also bounce off the bat and the player is able to control the motion of the bat. This is done with the keys surrounding the number 5. As each key is pressed the bat moves in the same relative direction as that key was to key number 5 (see figure 2). For example if the number 8 is pressed the bat moves straight up. If the number 1 is pressed the bat moves along a diagonal towards the lower left side. The bat will continue to move for as long as the key is pressed. The object of the game is to make the ball strike the grey area of the left wall before it strikes the grey area of the right wall.

Lines 5-100 of the program are initialization. A special input array is set up (more about this later) and boundary conditions are set. Lines 80-90 print the walls. If the walls were placed directly on the screen the right wall could be one column further right and both walls could be extended one line. For this example I chose the simplest method of initializing the screen.

The boundaries are memory locations 32768 thru 33727. The characters on the PET screen are related directly to the values in memory locations 32768 thru 33767. The screen fills from left to right and is 40 characters wide therefore poking a value into byte 32768 causes a character to appear in the upper leftmost (home) position, byte 32768 + 39 is the upper rightmost postion, byte 32768 + 40 is the leftmost position of the second line and so forth until byte 33767 which is the lower rightmost character position. Table 1 gives the values for each character to cause it to appear on the screen. Lines 25 & 30 set the conditions to keep the ball and bat from moving off the top or bottom of the screen. The grey areas of the walls provide the boundaries for the sides of the screen. The right grey area is actually the reverse field (rvs) of the left grey area therefore a peek (32768) would return a value of key & = 38 + 64 (for shift) = 102 while a peek (32768 + 39) would return 102 + 128 (for rvs) = 230. This provides an easy method of detecting when the sides of the screen are reached (and in this example an indication that the game is over).

To provide motion for the ball a horizontal and vertical displacement are used. This is so the ball can move in directions other than up, down, sideways, or diagonal. X0 is 32768 + the column and Y0 is the line number with 0 as top line. X and Y are increments which are added to X0 and Y0 to get the next position. (P1 is the next position while P2 is the current position). If the next position is beyond the top or bottom of the screen the direction of Y is reversed and the next position is set to the current position (lines 120-125) this provides a bounce The character on the screen at the next position is now checked (line 155). If this is equal to 35, the pound sign, (line 160) then the bat has struck the ball and it bounces off at a new angle. The magnitude of vector (S,Y) is fixed at 1 so that the ball cannot outrun the bat. If the next position has a screen value of 160 (32+128 for rvs blank) the white area of a wall was struck and the horizontal direction is reversed (line 180) but the new position is allowed to stand causing the ball to move into the wall. Lines 185-190 check for the winning or losing conditions. Finally in line 195 the next position is poked to the screen and the current position is blanked out (line 210). The current position is reset to the new position after looping to line 105 and the ball continues to move.

The bat is supposed to respond to the player and so a different movement scheme is used. The keyboard input routines supported by BASIC require one or more keys to be pressed and released for each input value to be received. This requires the player to tap at the keys like a woodpecker to control motion. To avoid this problem the program accesses byte 547 of the operating system working storage. When the interpreter is running the operating system places a unique value in this byte for each key that is pressed. (table 1 also gives these values, they are not the same as the screen character values). These values are then translated to a displacement for the bat.

The bat position is initialized and always kept at the actual address of the memory location which corresponds to the bats screen character position. A1 contains the next position while A2 contains the current position. In lines 35-45 an array E was set up with displacements stored at index values matching the values which may appear when any of the 8 keys surrounding number 5 is pressed. All other values of E are zero. By using the value at Peek (547) as an index to E the proper displacement for that key is obtained. For example when key number 2 is pressed, the value 18 appears at byte 547 and E(18)=40 which when added to the current position gives a next position one line lower(see lines 130-135) but if no key is pressed byte 547 contains 255 and since E(255)=0 the next position is the same as the current position and no motion takes place. The position is checked against the boundaries (line 140-150) and the screen is updated (lines 200-205). The program is now fast enough for the motion to appear continuous.

One drawback to this input scheme is that even though the keyboard buffer is not used to control the bat, it still fills up. Lines 310 and 320 show how the buffer had to be emptied before using the BASIC input routines again in line 370. When using the continuous keyboard input from a machine language routine it is important to leave the interrupt set to keyboard input or byte 547 may not get updated.

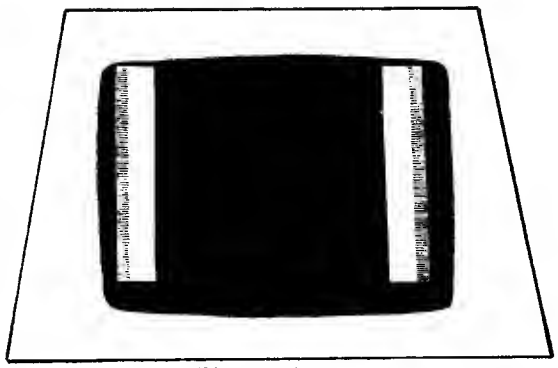


Figure 1
Showing the placement of the wall boundaries at the beginning of the game

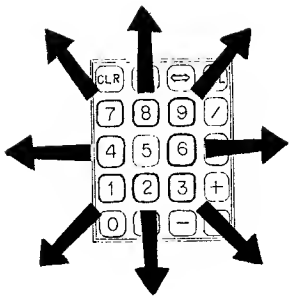


Figure 2

TABLE 1

KEY	SCREEN VALUE	KEYBOARD VAL (547)	KEY	SCREEN VALUE	KEYBOARD VAL (547)
@	0	15	blank	32	6
A	1	48	!	33	80
B	2	30	"	34	72
C	3	31	#	35	79
D	4	47	$	36	71
E	5	63	%	37	78
F	6	39	&	38	77
G	7	46	single quote '	39	70
H	8	38	(	40	76
I	9	53	)	41	68
J	10	45	*	42	33
K	11	37	+	43	17
L	12	44	comma ,	44	21
M	13	29	-	45	9
N	14	22	period .	46	2
O	15	60	/	47	49
P	16	52	0	48	10
Q	17	64	1	49	26
R	18	55	2	50	18
S	19	40	3	51	25
T	20	62	4	52	42
U	21	61	5	53	34
V	22	23	6	54	41
W	23	56	7	55	58
X	24	24	8	56	50
Y	25	54	9	57	57
Z	26	32	:	58	36

TABLE 1 (cont)

KEY	SCREEN VALUE	KEYBOARD VAL (547)	KEY	SCREEN VALUE	KEYBOARD VAL (547)
£	27	7	;	59	28
\	28	69	<	60	5
]	29	14	=	61	1
↑	30	59	>	62	12
←	31	7 5	?	63	20

The screen character values for a shift-key is the value of the key + 64. To get a reverse field (rvs) of a character (including shift-key characters) take the character value +128.

Additional keyboard values:

Home	74	
RVS	8	
STOP	4	(note pressing this key will still stop the program)
Up, down curser	66	
Sideways curser	73	
Del	65	

PROGRAM LISTING

```
 5 REM ** WALL BREAK **
10 REM ALAN K. CHRISTENSEN
15 REM     AUSTIN, TEXAS
20 DIM E(256)
25 T = 32768
30 B = 33727
35 E(58) = -41 : E(50) = -40 : E(57) = -39
40 E(42) = - 1 : E(41) =   1
45 E(26) =  39 : E(18) =  40 : E(25) =  41
50 X0 = 32788
55 Y0 = 11
60 A1 = 33148
65 P1 = 33188
70 X = RND(1) -.5 : Y = SQR(1-X*X)
75 ?"↑"
     clr
80 FOR I = 1 TO 25
85 ?"■↑~~"SPC(33)"~■"
     rvs
90 NEXT I
100 REM ** END OF INITIALIZATION **
105 A2 = A1 : P2 = P1
110 X0 = X0 + X : Y0 = Y0 + Y
115 P1 = X0 + 40 * INT(Y0)
120 IF P1 > B THEN Y = -Y : P1=P2
125 IF P1 < T THEN Y = -Y : P1=P2
130 I% = PEEK (547)
135 A1 = A1 + E(I%)
140 IF PEEK(A1) > 100 THEN A1=A2
145 IF A1 > B THEN A1=A2
150 IF A1 < T THEN A1=A2
155 P% = PEEK(P1)
160 IF P% <> 35 THEN 180
165 X = SGN(-X) * RND(1)
170 Y = SQR(1-X*X) * SGN(P2-A2)
175 P1 = P2
180 IF P%=160  THEN x=-X
185 IF P% = 102 THEN 300
190 IF P% = 230 THEN 400
195 POKE P1,87
200 POKE A1,35
205 IF A1<>A2 THEN POKE A2,32
210 IF P1<>P2 THEN POKE P2,32
215 GOTO 105
```

```
300 REM *** WINNER ***
310 GET A$
320 IF A$ <> "" THEN 310
330 ?" ↑"SPC(12)"↑CONGRATULATIONS"
      ↑home    ↑rvs
340 FOR I = 1 TO 100 : NEXT I
350 ?" ↑"SPC(12)"CONGRATULATIONS"
      ↑home
360 FOR I = 1 TO 100 : NEXT I
370 GET A$
380 IF A$ = "" THEN 330
390 GOTO 50
400 REM *** LOSER ***
410 GET A$
420 IF A$ <> "" THEN 410
430 ?" ↑"SPC(12)"↓SORRY↓ ↑TRY↓ ↓AGAIN"
      ↑home   rvs↑   off↑ rvs off↑ rvs
440 FOR I = 1 TO 100 : NEXT I
450 ?" ↑"SPC(12)"SORRY TRY AGAIN"
      ↑home
460 FOR I = 1 TO 100 : NEXT I
470 GET A$
480 IF A$ = "" THEN 430
490 GO TO 50
500 END
```

THE SIEVE OF ERATOSTHENES

Gary J. Bullard
1722 S. Carson, #1502
Tulsa, OK 74119

Over 2000 years ago, a Greek geographer-astronomer named Eratosthenes devised a way of finding prime numbers that is still the most effective known. He simply started with the number 2 and crossed out all multiples of 2. Then he took the next number that had not yet been crossed out (3) and proceeded to cross out all multiples of it. And so on until he had found all the prime numbers he was interested in. This method of finding prime numbers is called a "sieve" because the prime numbers fall through the holes created by crossing out all the non-prime numbers.

So what? Well, this gives rise to an interesting program for the PET. Picture the 1000 character positions on your PET's screen as the numbers 1 to 1000. Now cross out all the positions that represent non-prime numbers. What you have left is a strange pattern that would make an interesting bathroom tile arrangement. It also shows the placement of the prime numbers occurring between 1 and 1000.

```
10 PRINT CHR$(147);
20 DIM A(200)
```

Line 10 simply clears the screen. PET users can use the CLR function rather than the CHR$(147). Line 20 reserves storage for the prime numbers we will extract later. (There are more prime numbers than you might think in the range of 1 to 1000.)

```
90 FOR N=2 TO 35
95 IF PEEK(N↑32767)=102 THEN 130
100 FOR X=32767↑(2*N) TO 33767 STEP N
110 POKE X,102
120 NEXT X
130 NEXT N
```

This double loop is the meat of our program. We only loop 34 times (2 to 35) because it is only necessary to test for multiples of primes up to the square root of your limit - in this case SQR(1000)=31↑. (I added a couple for good measure). Line 95 checks the screen to see if our next potential prime has already been crossed out. Line 100 does the stepping across the screen, and line 110 does the "crossing out." Note that the PET's screen is actually addressable memory beginning at 32768(10).

```
200 N=1
210 FOR X=1 TO 1000
220 Z=PEEK(32767↑X)
240 IF Z=32 THEN POKE(32767↑X),81:A(N)=X:N=N↑1
250 NEXT X
```

Now that we have crossed out all the non-primes, it is time to see what was left. This loop examines the screen to find the spaces. The index "X" will tell us what character we are looking at and the counter "N" will give us the next empty space in our table to store the prime number. Line 200 sets the table pointer to 1. Lines 210-250 is the loop that examines the screen. Line 220 looks at the current character position and puts its value in Z. In this case, the value will be 102 if it is a crossed out position, and 32 if it has not been crossed out. Line 240 then tests the value of Z and either ignores it if it has been crossed out or saves it in our table if it is prime.

```
300 GET A$:IF A$=" " THEN 300
```

This line simply causes the PET to pause while you admire its handiwork. When you are ready to see a list of the prime numbers, press any key.

```
400 PRINT CHR$(147);
410 FOR X=1 TO 200
420 IF A(X)=0 THEN STOP
430 PRINT A(X);
440 NEXT X
```

Line 400 clears the screen again. Lines 410-440 recovers our prime numbers from the table and prints them. When the table returns a zero, then we are finished, and the program will stop (line 420).

```
999 END
```

I hope you enjoyed this little bit of updated history. I'm sure old Eratosthenes would have been very happy to have had a PET to play with, but even 2000 years later he is not out of date.

INSIDE PET BASIC

Jim Butterfield
14 Brooklyn Avenue
Toronto, Ontario
Canada M4M 2X5

PET BASIC is pretty good: fast, powerful, and flexible. Most of the time you can write programs without ever needing to know what's inside. But there are a few handy things that you can't do without "dissecting" BASIC. Let's take a couple of examples. Suppose you want to look through a big program for some reason. You might have a small bug: say a variable, X4, ends up with a wrong value, and you want to find out why. You could list the program, a screenful at a time, looking for every time X4 is used; but eye fatigue starts to set in. Wouldn't it be nice to have a utility program to do the scanning for you?

Program FIND

Program FIND will do the job for you. To write such a program, though, we need to know how BASIC is built. The first line of your BASIC program starts at address 1025 (or 0401 hexadecimal). That's where we must start our search. Each BASIC line will have the following format: The first two locations contain a pointer to the next line of BASIC; or if they contain zeros, there is no next line and this is the end of your program. The next two locations contain the BASIC line number. After that (starting at the fifth location) we have the BASIC line itself. It's mostly in ASCII code, but keywords such as FOR, PRINT, or SQR are stored as special codes known as "tokens". At the end of the line we'll find the value zero.

How do we use this information to scan BASIC for a given expression? First, we set our address, A, to 1025; that's where BASIC starts. Next, we skip over the first four bytes (pointer and line number) and search from A+4 to the end of the BASIC line. We'll recognize the end-of-line by the zero at the end. If we find the expression we want, we can output the line number by obtaining it from A+2 and A+3. It's in binary, so we use the expression 256*PEEK(A+3)+PEEK(A+2) - printing this value will print the line number.

When we reach the end of the BASIC line, we must go to the next line, of course. It will be right behind the zero that marked the end of our previous line; or we can use the pointer to jump ahead with A=256*PEEK(A+1)+PEEK(A). If the pointer is zero, we know that we have come to the end of the BASIC program and can stop.

Program RESEQUENCE

Let's move on to something more complicated. Suppose you want to renumber your BASIC program. Since we know how the line numbers are stored in BASIC, it seems easy; we'll just change them to the new values. There is a hitch, however. What happens if your program contains a GOTO 300 statement - and now line 300 is renumbered so that it becomes line 380? Problems - that's what happens.

What we must do is search out all the GOTOs and GOSUBs, including those included in ON.. statements, and be ready to change the old line numbers to new ones. One way of doing this is to build a table of "old" addresses, match them with the "new" line numbers, and then correct them after renumbering has been accomplished. To help make things more complicated, we have two different ways of using the THEN statement. If we have a line such as IF J=12 THEN Y=2, there is no line number reference to correct. On the other hand, if we have IF J=12 THEN 530, we must be ready to fix up 530, replacing it with a new line number if necessary.

More difficulties: if we have a statement which says, for example, GOTO 5, and with the renumbering we want to change it to GOTO 100, we won't have space! And making space isn't that easy: you may recall that the lines of BASIC are "chained" together with pointers; if we lengthen a BASIC line, all the pointers will need to be fixed up! This last problem is too tough to resolve in a simple manner - let's sidestep it by printing a warning notice if it should occur.

How do we approach this job? We separate the program into three phases. Phase 1 looks through the program for line number references and builds a table. Phase 2 does the actual renumbering (the easiest part of the whole job). Phase 3 looks through the program again and corrects the line number references. How do we look through the program? The same way as with program FIND. We're looking for three keywords: GOTO (token 137), GOSUB (141) and THEN (167). Sometimes we'll also allow a comma (44) so that statements such as ON X GOTO 100,200,300 will be allowed. You'll see this testing for tokens on line 60220 of RESEQUENCE.

If we find one of these keywords, we must convert the following ASCII numbers into a value V corresponding to the line number. During Phase 1, we build these line numbers into a table at 60090. Phase 2 is a snap. In lines 60030-60040 we change the line number and then check to see if the old number was in table V%. IF so, we fill in the cross-reference. Phase 3 is the long one. We must repeat the search of Phase 1. Then, in 60110 to 60150 we must build the new line number (in ASCII) and insert it - with appropriate tests and warning notices.

Making Them Work

Both FIND and RESEQUENCE are written in BASIC. That means that they will have to reside in PET's memory along with the programs they are dealing with. RESEQUENCE is constructed so that it doesn't renumber itself, of course; and FIND will examine itself, reporting any occurences of the search string. Another problem arises, however: how can you get two programs into the PET at the same time? We need to load either FIND or RESEQUENCE together with the program that is being processed. A normal PET load wipes out the old program when a new one is loaded. You could always add FIND or RESEQUENCE by entering it at the keyboard; this would add the utility program to the existing program in memory. But such a procedure is lengthy and it would be easy for errors to creep in. There must be a better way. One good way is to use the screen as a "holding buffer". You could load program FIND, and list it onto the screen. Then load the program you want to search. FIND will be wiped out of memory, but it's still on the screen - so you

can move the cursor back to displayed line 9000, and hit RETURN eight times. FIND will be restored to memory, where it now shares space with the program to be scanned. This doesn't work too well with a longer program like RESEQUENCE, however. The program is too big to fit on the screen - much too big. There must be another even better way. Larry Tessler of Sphinx opened the door with his program UNLIST, which made true program merging possible for the first time. Since this breakthrough, an even better method has been devised by Brad Templeton of Toronto.

UNLIST - A Procedure for Merging Programs

Here's how it works. Be sure to follow the instructions carefully and exactly. Prepare the programs you will want to merge in the following manner. Load the program. Place a blank tape into your cassette unit. Now type:

```
OPEN 1,1,1:CMD 1:LIST
```

When the tape stops, type:

```
PRINT#1:CLOSE1
```

and your merge tape is ready. At a later time, when you want to merge the program, here's what to do. First, mount the merge tape you previously prepared and type OPEN 1. Now clear the screen, give exactly four cursor downs, and type the following, but DO NO HIT RETURN:

```
POKE611,1:POKE525,1:POKE527,13:?"h"
```

(h is cursor home; shows as reverse S). Don't hit return: press cursor home and give six (6) cursor downs. Now type exactly the same line (two lines below the first line) and then hit RETURN. The tape will more; the merge will take place; and finally, an error notice will print between the two lines. Stop the tape if it's still going, and then type CLOSE1. Miraculously the merge has taken place!

How does it work? It's a little complex; but if I hinted that POKE 611,1 transfers control away from the PET's keyboard to the cassette tape, you'd have part of the story. And if I mentioned that poking 525 and 527 simulates a RETURN key being hit, you'd have another part. But, you don't need to know what makes it work in order to use it. Use it; benefit from it; and enjoy it.

FIND for PET

Need to search a program for an express, a variable, or a keyword? Slip program FIND in behind your program (it's not very long) - then insert a line 1 to say what to search for ... and the job's done. Every line in memory which contains the same expression as line 1 will be reported. This includes line 1 itself, of course, and any lines in program FIND ... as well as the program you're searching. The program is listed here spaced out for readability - close in the spaces when you input to save space.

```
9000 A=1025 : X=PEEK(1029) FOR J=1 TO 1E3 : FOR
     K=A+4 TO A+83
9001 P=PEEK(K) : IF P=X THEN GOSUB 9005
9002 IF P<>0 THEN NEXT K
9003 A=256*PEEK(A+1)+PEEK(A) : IF A>0 THEN
     NEXT J
9004 STOP
9005 FOR L=1 TO 80 : Y=PEEK(1029+L) : FI Y=0
     THEN ? 256*PEEK(A+3)+PEEK(A+2); : RETURN
9006 IF Y=PEEK(K+L) THEN NEXT L
9007 RETURN
```

Example: to find all FOR statements in a program; insert FIND (above) and then insert line 1

```
1 FOR
```

Now invoke FIND with RUN 9000. The program will print 1 followed by any program lines containing FOR followed by 9000 9000 9005 (9000 prints twice because it contains two FORs).

FOR is a keyword, and doesn't store as three separate characters, so you wouldn't find it if you searched for characters FO. This can be handy: if you were looking for variable F you wouldn't get all the FORs printed.

Modifications: if you squeezed P=0 just ahead of RETURN on line 9005 (it's a tight squeeze) a line number would print only once even when it had multiple matches; you might or might not want this feature.

IMPORTANT: Don't forget to wipe out line 1 and program FIND when you're finished with them.

RESEQUENCE for PET

```
60000 END
60010 TO= : DIM V$(100),W$(100) : GOSUB 60160 :
      FOR R=1 TO 1E3 : GOSUB 60210
60020 IF G THEN GOSUB 60090 : NEXT R
60030 GOSUB 60160 : FOR R=1 TO 1E3 : N=INT
      (M/256) : POKE A-1,M-N*256
60040 POKE A,N : V=L : GOSUB 60070 : W$(J)=M :
      GOSUB 60170 : IF G THEN NEXT R
60050 GOSUB 60160 : FOR R=1 TO 1E3 : GOSUB 60210
      : IF G THEN GOSUB 60110 : NEXT R
60060 ?"*END*" : END
60070 J=0 : IF T<>0 THEN FOR J=1 TO T : IF V$(J)
      <> V THEN NEXT J : J = 0
60080 RETURN
60090 IF V<>0 THEN GOSUB 60070 : IF J=0 THEN T=
      T+1 : V$(T)=V
60100 RETURN
60110 GOSUB 60070 : IF J=0 THEN RETURN
60120 W=W$(J) ; IF W=0 THEN ?"GO";"L";L;"?":
      RETURN
60130 FOR D=A TO B+1 STEP-1 : X=INT(W/10) :
      Y=W-10*X+48 : IF W=0 THEN Y=32
60140 POKE D,Y : W=X : NEXT D : IF W=0 THEN
      RETURN
60150 ?"INSERT";W$(J);"L";L : RETURN
60160 F=1025 : M=90
60170 A=F : M=M+10
60180 F=PEEK(A)+PEEK(A+1)*256 : L=PEEK(A+2)+
      PEEK(A+3)*256 : A=A+3 : G=L<6E4
60190 RETURN
60200 S=0
60210 V=0 : A=A+1 : B=A : C=PEEK(A) : IF C=0
      THEN GOSUB 60170 : ON G+2 GOTO 60210,60190
60220 IF C<>137 AND C<>141 AND C <>167 AND C<>S
      GOTO 60200
```

```
60230 A=A+1 : C=PEEK(A)-48 : IF C=-16 GOTO 60230
60240 IF C>=0 AND C<9 THEN V=V*10+C : GOTO 60230
60250 S+44 : A=A-1 : RETURN
```

RESEQUENCE can sit quietly behind your program. When you say RUN 60010, your program is renumbered. RESEQUENCE gives error notices if:

A. a GOTO or GOSUB statement wants to go to a non-existant line;
B. there isn't enough room for a new (higher) line number.

In both cases you're given the (new) line number where this happens. RESEQUENCE doesn't run fast (allow about a second per line, more for large programs), but it's dependable and very useful.

Program comments: Line 6000 stops the user program if it gets here. Lines 60010-60020 extract all GOTO, GOSUB, and THEN references and build them into a table. Lines 60030-60040 renumber all lines, and cross-references the table if needed. Line 60050 updates all line references.

Subroutines: 60070 looks for an entry in the line number table. 60090 inserts a new entry into the table. 60110 revises a line number reference. 60160 starts a new scan of the user program; 60170 continues the scan with the next line. 60210 scans the user program for GOTOs, etc.; value S is used to accomodate ON A GOTO ... type situations.

Author's Notes:

Reader questions suggest that the following additional information may be useful:

UNLIST procedure: when you mount the previously prepared merge tape and type OPEN 1...

-follow this statement with a carriage return in the normal way;
-PET will want to read tape; press PLAY as requested. Tape will move, and eventually PET will report FOUND. Now clear the screen and continue with the POKE 611,1 procedure.

RESEQUENCE: the program as written will handle line numbers up to 32767, which gives lots of scope in program-writing. If you need to handle higher line numbers, change V%(to V(throughout the program.

Using V%(saves space in memory, since integer arrays are stored very efficiently. However, the highest integer allowed is 32767, so that higher line numbers won't fit.

It's probably obvious that the user program must have all its lines below 60000 - since RESEQUENCE itself starts at that point.

GENERAL

Manufacturers of 6502 Microcomputers covered in Best of MICRO Volume 2

APPLE COMPUTER INC. 10260 Bandley Drive, Cupertino, CA 95014	APPLE II 408/996-1010
COMMODORE BUSINESS MACHINES, INC. 330 Scott Blvd., Santa Clara, CA 95050	KIM-1 & PET 408/727-1130
OHIO SCIENTIFIC 1333 S. Chillicothe Road, Aurora, OH 44202	SUPERBOARD & Others 216/562-3101
ROCKWELL INTERNATIONAL Microelectronic Devices, P.O. Box 3669, Anaheim, CA 92803	AIM 65 714/632-0950
SYNERTEK SYSTEMS CORPORATION P.O. Box 552, Santa Clara, CA 95052	SYM-1 408/988-5600

6502 INTERFACING FOR BEGINNERS; THE CONTROL SIGNALS

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

By now your breadboard should look like a rat's nest so we shall add just a few more wires. So far you have used several decoding chips to produce device select pulses (also called chip selects, port selects, etc.) These pulses activate a particular I/O port, memory chip, PIA device, interval timer or another microcomputer component. Almost all of these components must "know" more than that they have been addressed. They must know if the microprocessor is going to READ data from them or WRITE to them. The R/W control line coming from the R/W pin on the 6502 provides this information. It is at logic 1 for a READ (typically LDA XXXX) and at logic 0 for a WRITE (typically STA XXXX).

If you have ever tried to wrap your mind around timing diagrams for microcomputer systems you soon realize that system timing is also important. Suppose that a memory chip is selected by a device select pulse. A 21L02 chip, after being selected, must decode the lowest 10 address lines itself to decide which of its 1024 flip-flops will become the output data. This takes time, so the data at the output pin is not ready instantaneously. The 6502 simply waits for a specified amount of time, and at the end of this period it reads the information on the data bus. If the access time of the chip is too long, the 6502 will read garbage; otherwise it will get valid data.

Likewise, during a WRITE cycle, the microprocessor brings the R/W line to logic 0, selects the device which is to receive the data, and at the end of a cycle it signals the divice to read the data which the 6502 has put on the data bus. The signal which successfully concludes both a READ and a WRITE instruction is the so-called phase-two clock signal symbolized by 0_2. In particular, it is the trailing edge (positive to zero transition) of this signal which is used.

All the timing for the microcomputer is done by the crystal oscillator on the microcomputer board and the clock circuitry on the microprocessor itself. A clock frequency of 1 MHz produces a machine cycle of 1 microsecond in duration. Near the beginning of the cycle the address lines change to select the divice which was addressed, and the R/W goes to logic 1 or logic 0 depending on whether a READ or a WRITE was requested. If a READ was requested, some device in the system responds by putting data on the data bus. Typically this happens during the second half of the cycle when 0_2 is at logic 1. Finally, at the end of the cycle, but before the address lines or the R/W line have changed, 0_2 changes from logic 1 to logic 0, clocking the data into the 6502. The same kinds of things happen during a WRITE cycle, except that now the external device uses the trailing edge of the 0_2 signal to clock the data, while the 6502 puts the data on the bus at a slightly earlier time in the cycle. For details refer to the 6502 HARDWARE MANUAL.

The circuits you have built so far, together with a few more chips, will demonstrate the effect of the control signals. Refer to Figure 1 of the last installment of this column (MICRO, Issue 6, p. 30), and to Figure 1 of this issue. You will see the LS145 and the LS138 have not been changed too much, in fact all of the connections to the LS145 should stay the same. The device select pulse from the LS145 goes to G2A as before, but another signal goes to G2B in the new Figure 1. For the moment disregard the lower LS138 and LS367 in Figure 1 of this issue. The new signal to G2B of the LS138 is our WRITE signal. It is produced by NANDING the $\overline{R/W}$ signal with 0_2 and it is an active-low signal. On the KIM-1 it is called RAM-R/W and is available on the expansion connector. Most other 6502 systems will very likely also have a RAM-R/W signal.

Its effect in Figure 1 is to inhibit the devic select pulse from the LS138 whenever the R/W line is high (during all READ instructions), but to allow the device select pulse to occur when the R/W line is low and 0_2 is high. Thus, the top LS138 in Figure 1 selects output ports only, and the device select pulse from it terminates on the trailing edge of the 0_2, producing a logic 0 to logic 1 transition simultaneously (almost) with 0_2. This pulse is inverted by the LS04. Consequently, a WRITE instruction produces a positive pulse at the G inputs of the LS75 whose duration is about 1/2 microsecond and whose trailing edge coincides with 0_2.

The 74LS75 is a 4-bit bistable latch whose Q outputs follow the D (data) inputs only when the G inputs are at logic 1, in other words during the device select pulse from the LS04 inverter. The trailing edge of this pulse latches the Q outputs to the value of the D inputs during the device select pulse. If you had a great deal of trouble following this, you may want to check the reverse side of this page to make sure there is nothing valuable on it and then destroy this by burning or shredding! Otherwise proceed to to the experiment below.

Connect the circuit shown in Figure 1, omitting for the time being the lower LS138 and the LS367. You can also omit the connection of address line A3 to G1 on the top LS138 if G1 is connected to +5V as was indicated in the last issue. In other words, simply add the LS04 and the LS75 to your circuit of the last issue. The RAM-R/W signal must also be generated if your 6502 board does not have one. Simply use one inverter on the LS04 to invert the R/W signal to R/W, then NAND it with the 0 , and run the output of the NAND gate to the G2B pin on the LS138.

The address of the device is 800F if the connections are made as shown in the figure. If other pins on either the LS145 and/or the LS138 are changed the address will be different. The switches shown connected to the D inputs may be implemented with a DIP switch or jumper wires. An open switch corresponds to a logic 1 while a closed switch is logic 0. Set the 4 switches to any combination then load and run the following program:

```
0200  8D 0F 80      STA DSF.
```

The LEDs should indicate the state of the switches. If you add the statements

```
0203  4C 00 02      JMP START
```

then you should be able to change the switches and the LEDs will follow the switches. Try substituting an AD 0F 80 (LDA DSF) for the 8D 0F 80 instruction. Nothing should happen, even though the same address is being selected, because on LDA instruction the R/W line is high, inhibiting the LS138 from producing a device select. Fin-

ally, connect the data lines D0-3 from the 6502 to the D-inputs of the LS75, making very sure that the LS145 is de-selecting other locations. On the KIM-1 this means that pin 1 of the LS145 is connected to pin K on the application connector and pin 9 of the LS 145 is connected to pin J. The appropriate pull-up resistors must also be added. With the data lines connected run the following program:

```
0200  A9 04     LDAIM $04
0202  8D 0F 80    STA DSF.
```

Play around with different numbers in LDAIM instruction and explain your results. If nothing seems to make sense, it may be that your data lines need to be buffered, a topic we will take up next issue. If your results make sense you will have discovered that we have configured a 4-bit output port whose address is 800F. Adding another LS75 to connect to data lines D4-D7 and whose G connections also go to the output of the LS04 will give an 8-bit output port. Seven other output ports, addresses 8008 through 800E, could be added using the other device select signals from the LS138, LS04 inverters, and LS75 latches.

If you want to make an input port wire the circuit for the lower LS138 in Figure 1. If you don't have much more room on your circuit board you might want to simply reconnect the upper LS-138 to become the lower LS138. A couple of connections do the trick. Set the switches to anything you like and run the program below.

KIM-1 users should see the hex equivalent of the switch settings appear in the right-most digit on the display. Owners of other systems can omit the last two lines of the program, stop it, and examine the location 00F9 to see that the lowest four bits agree with the switch settings. Experiment with other switch settings to make sure that everything is operating correctly.

The completed circuit of Figure 1 gives one 4-bit output port (provided the data lines are connected to the D inputs of the LS 75) and one 4-bit input port, addresses 800F and 8007 respectively. These two ports are easily expanded (two more chips) to become 8-bit ports. Likewise the circuit of Figure 1 could be expanded to give a total of eight 8-bit input ports and eight 8-bit output ports.

Next issue we will look at a slightly different input port, and we will look in more detail into three-state devices and the data bus. You may want to keep your circuit together until then.

```
0200  AD 07 80   START   LDA DS7      Read input port data
0203  85 F9              STA DISP     and store it in location 00F9.
0205  20 1F 1F            JSR SCANDS   Jump to KIM display subroutine.
0208  4C 00 02            JMP START    Repeat program.
```

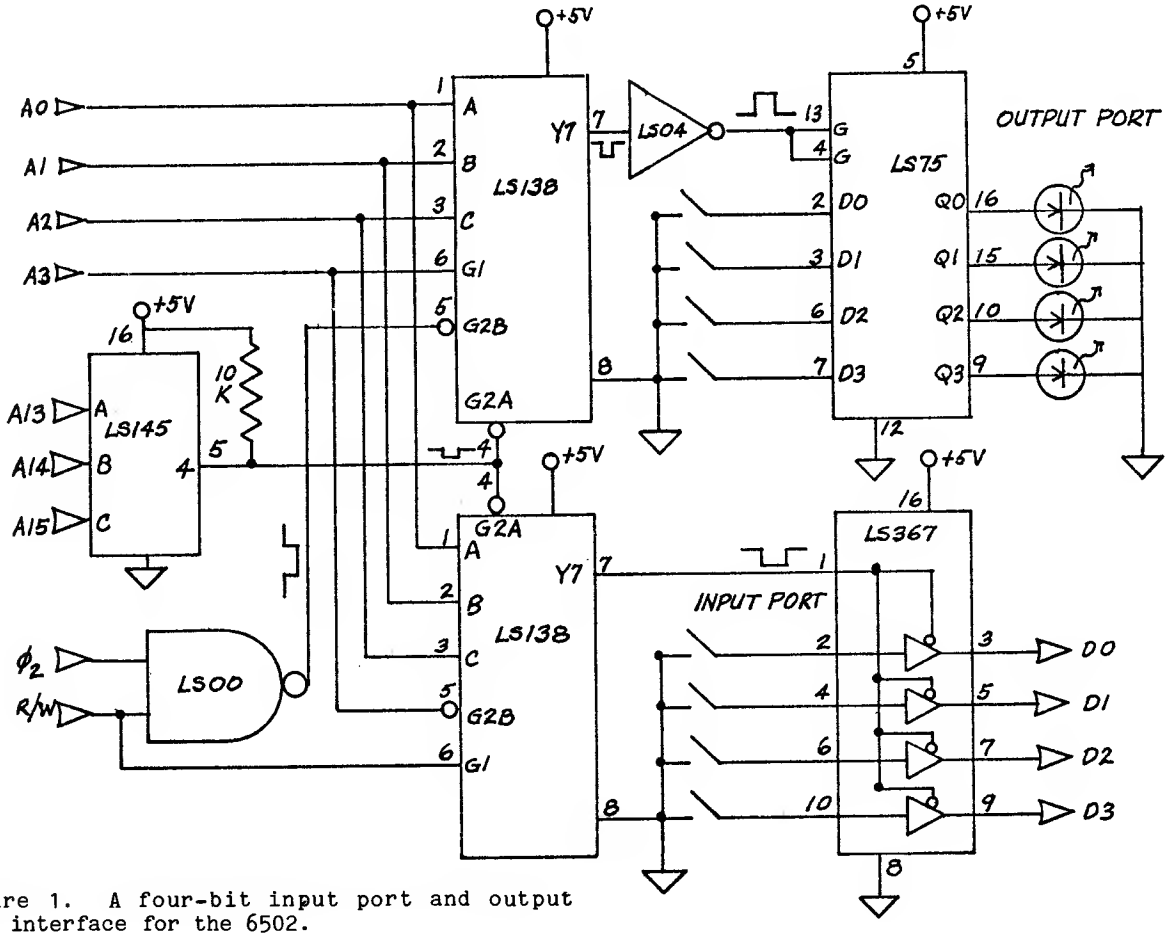


Figure 1. A four-bit input port and output port interface for the 6502.

6502 INTERFACING FOR BEGINNERS:

BUFFERING THE BUSSES

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

The address bus is the set of 16 conducting lines interconnecting the 6502 and numerous other integrated circuits in the computer system such as memory chips, PIAs, decoding circuits, etc. On my 8K memory board the address bus is connected to 64 memory chips. The address bus carries the addressing information from the 6502 to the other components in the system. It is, consequently, a one-way bus, in contrast to the data bus which carries signals both ways.

The control bus is a set of conductors which connect the 6502 control signals (Ø , R/W, SYNC, RST, NMI, IRQ, RDY, and SO) with the other components in the microcomputer system. Some control signals originate in the 6502 and these are bussed to the system. Other control signals e.g. NMI and IRQ, originate somewhere in the system and are bussed to the 6502. None of the control signals use a bi-directional bus like the data bus.

Finally, the data bus is a set of 8 conductors connecting the 6502 and the other devices in the system. It presents a special problem because it is required to carry information two ways, hence the name "bi-directional data bus." On a WRITE command the data bus carries an 8-bit word (one bit on each line) from the 6502 to a memory location, while on a READ command the data bus carries information from a memory location to the 6502. On my 8K memory board each data line is connected to 8 memory chips.

WHY BUFFER?

There are two reasons for buffering uni-directional busses like the address bus and the control bus:

1. The address and control pins on the 6502 are rated to drive one standard TTL load. In any but the simplest computer system there will be heavier loading than this.

2. Every conductor including those which make up the busses has some capacitance. Capacitors require time to charge and discharge and "distort" rapidly changing waveshapes. Buffer chips can drive a much larger capacitance than the 6502, and consequently may be inserted to preserve the integrity of the waveshapes of the signals.

In addition, the data bus requires a special kind of buffer. Recall that the microprocessor is capable of reading data from any of 65,536 devices. But only one at a time, please. All the others should act as if they are not there, which means they should be disabled somehow. If two devices are both attached to a data pin, one trying to raise it to logic 1 and the other trying to lower it to logic 0, not even a prophet can predict the result. The third reason for buffering applies only to bi-directional busses and may be summarized:

3. Buffers must be capable of isolating the bus from all of the devices on the bus except those which have been addressed (for example, the 6502 and an input port) and between which data is being transmitted.

BUFFER/DRIVER CHIPS

We mentioned earlier that all the bus pins on the 6502 are rated to drive one standard 7400 series TTL load. This means that you could connect about four 74LS00 series chips to a bus line, but if you tried to hang additional chips on these lines the circuit would probably not operate. For the address bus and the control bus the solution is to connect the 6502 pins directly to two 7404 inverters (or 74LS04's). A 7404 can drive 10 standard TTL loads and about 40 LS loads, while a 74LS04 can drive 20 74LS00 series loads. This should provide adaquate drive for most systems, provided the bus length is not to great. If you have a KIM-1 schematic you will note that both R/W and Ø are buffered in this manner, but that none of the address lines are buffered because the KIM-1 system is small enough to not require buffering. However if you expand, the address lines will also require buffering. As an example, see KIM USER NOTES, Issue #7,8 where Jim Pollock gives a KIM to S-100 circuit.

There are other chips called Bus Buffers/Drivers which can be used either on uni-directional busses or the bi-directional data bus. They come in packages of four (quad), six (hex) or eight (octal) buffer/drivers to a chip. If you want to look up the specs on some of these chips here are a few of the more popular ones.

74LS125 quad	DM8093 quad
74LS126 quad	DM8094 quad
LS367 hex	DM8097 hex
8T97 hex	81LS97 octal

All of these except the 81LS97 are readily available (Jameco, Godbout, Jade, etc.). The only place I have been able to find 81LS97's is Hamilton-Avnet. They are a bit more expensive and come in a 20 pin package, but they are nice because they can handle eight lines. Note that we have already used the 74LS367 to buffer address lines. Refer to the last several columns of this feature.

The truth table and logic symbol for a typical buffer/driver are given in Figure 1. Carefully focus your beady eyes on the function of the G (gate) input.

Note that when G is low the output follows the input logic level. The device is then doing its thing, namely driving the particular bus line to which it is attached. The inversion circle indicates that the buffer/driver is active (works) when the gate signal is a logic 0. Some buffers have no inversion circles, and they will be active when the gate is at logic 1. Perhaps the most important feature is the third state of the output in the truth table, which we have labelled "disabled." When the gate is high the device behaves as if it were disconnected from the bus, that is just as if a switch in series with output were opened. This property is the reason for calling these devices "three-state buffer/drivers" or" or "TRI-STATE buffer/drivers." (TRI-STATE is a trademark of National Semiconductor.)

Figure 2 shows how an LS 125 might be used on the bi-directional data bus. Only two bus lines are shown for simplicity. During a WRITE instruction the R/W line is low, enabling the buffers which drive the signals from the 6502 to the external devices. The other buffers which drive the 6502 are disabled. Analyze what would happen if they weren't disabled! During a READ instruction the R/W line is high, it is inverted by the LS04, and it enables the buffers driving the signal from the external devices to the 6502.

The scheme shown in Figure 2 is not the only possibility. For example, the S-100 bus would not have pins 3 and 5 connected, nor pins 8 and 12 connected. Instead, the data bus is divided into two separate busses at this point. The bus lines connected to pins 3 and 8 become a "data out" bus, while the lines connected to pins 5 and 12 become a "data in" bus. I am not aware of all of the advantages and disadvantages of this scheme, so we will not pursue it further.

AN EXPERIMENT

Connect an LS125 as shown in Figure 3. Note that RESET will very likely cause all the LEDs to light. Now run the following program:

```
0000  4C 00 00          START   JMMP START
```

This is an infinite loop. Do not try to relocate the program or the experiment may not work. You should observe that the LEDs on D0 and D1 are off while the other two are one. Can you explain why before I do?

Analyzed by clock cycles the activity on the data bus may be summarized as follows:

The LEDs connected to D3 and D2 get a pulse once every three clock cycles, which the eye interprets as a continuous glow. Now connect the gates (pins 1,4,10,13) to +5V instead of ground. None of the LEDs light. Why?

AN OBSERVATION

Refer to Figure 1 in the "INTERFACING...." column in MICRO #7. The input port illustrates how a buffer/driver isolates the data bus. Note that the device select pulse is connected to the gate of the LS367. Thus, only when the address lines select the input port and the 6502 is in the READ state does the LS367 control the data lines. Otherwise it is disabled and the 6502 gets its data elsewhere.

The output port of the same circuit illustrates another point. Suppose we had say eight output ports. Data lines D0-D7 would each have eight LS inputs hanging on them, and the 6502 would probably be unable to drive them. The solution would be to buffer the data lines from the 6502 to the output ports. In this case one would probably connect the R/W line to the buffer/driver gates.

AN APPLICATION

Again refer to Figure 1 in this column in MICRO # 7. Recall that the data lines were to be connected to the D inputs of the LS75 to complete the output port, replacing the switch. A complete 8-bit output circuit, with buffering, is shown in Figure 4. The device select circuitry is not repeated here. Up to eight output ports can be implemented using the device select pulses from the LS138. All you have to have are LS 75s. The buffering shown in Figure 4 would be more than adaquate for eight ports.

The 8-bit port with LEDs attached can be used as a debugging tool among other things. At a point in a program where you suspect trouble, and want to see the STATUS REGISTER for example, put a BREAK command. The last thing on the stack after a break is the status register contents. So, the interrupt vector should point to a program which pulls the last word off the stack and loads it at the address of the output port, STA $800F. A little panel could be made which indicates LED goes with which flag.

The scheme just mentioned can obviously be varied to indicate the contents of any of the important registers. One could get very elegant and use four ports to indicate X, Y, accumulator and status register simultaneously. Better yet, use the information you have learned to display the contents of X,Y,A, and P while the computer is in the single-step mode.

What's next? I hope to go into a keyboard input port in a little more detail, then look at a memory interface, unless I get some other ideas that is. Anyway, you ought to step out from among the trees to get a look at the forest by taking a long and studied look at Figure 1.1 of the MOS TECHNOLOGY HARDWARE MANUAL, the first figure in the book. A lot of the ideas we have been discussing are summarized there in a diagram of the microcomputer system as a whole.

Parts list of components used for the experiments.

1	AP Circuit Board (holds 8, 16-pin DIPs)
1	coil, #22 wire
8	LEDs
1	Edge connector for KIM-1
1	74LS145
2	74LS138
1	74LS04
1	74LS367
2	74LS75
2	74LS125
1	74LS76
2	4.7K to 10K resistors
2	DIP switches

An LS125 and LS04 in a bi-directional data bus buffering circuit. Only two data lines are shown buffered. Four LS125s would be required for all eight data lines. In this scheme the "write" buffers and "read" buffers are alternately disabled by the R/W line. Sometimes they are also disabled by device select pulses.

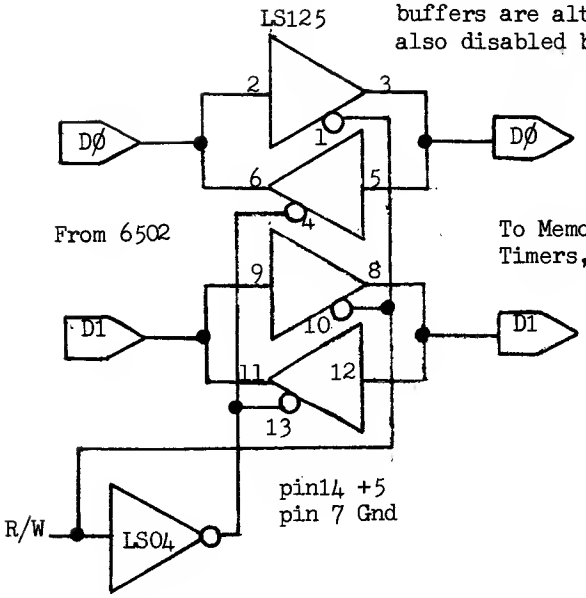


Figure 2.

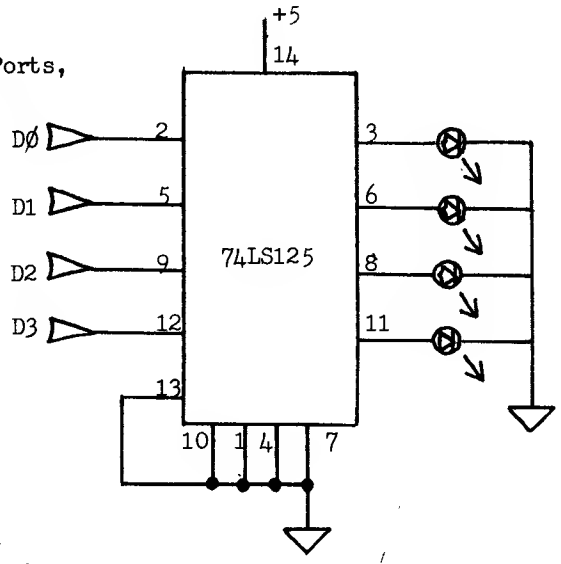


Figure 3.

Circuit to demonstrate data bus buffering. See text for details.

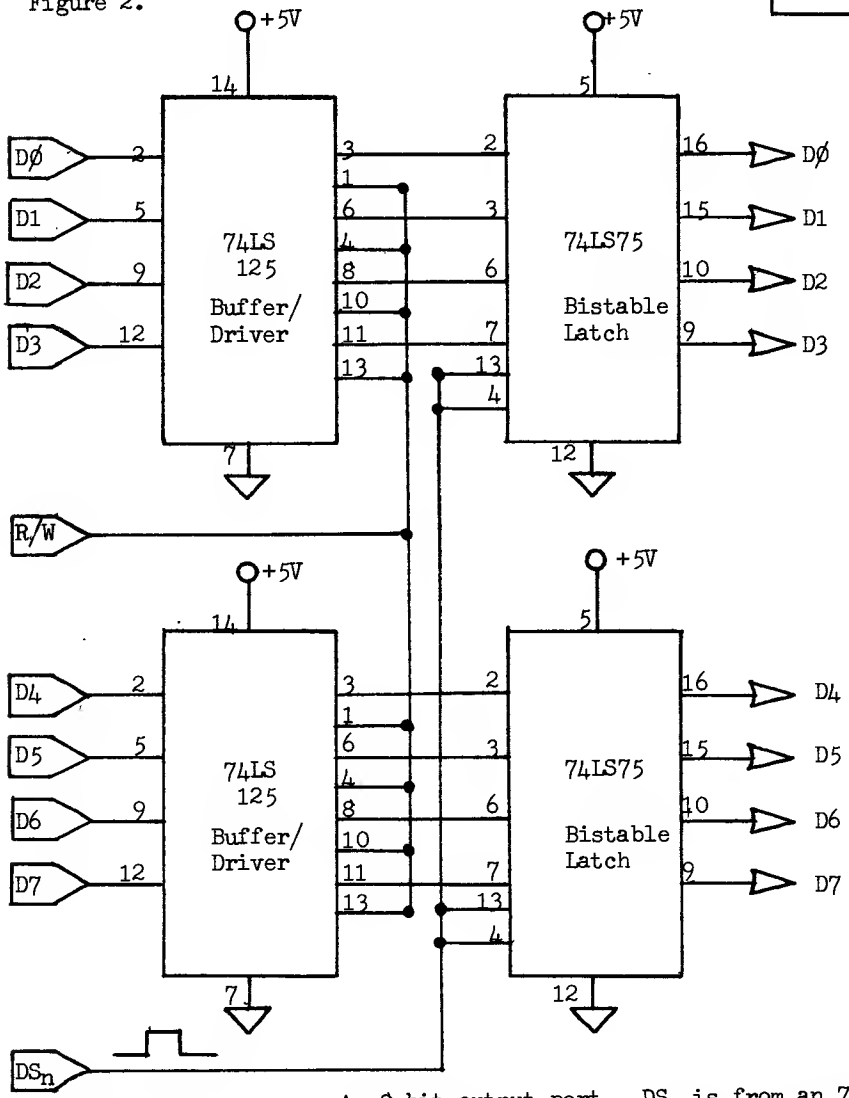


Figure 4.

An 8-bit output port. DS_n is from an 74LS138 and LS04 inverter. The buffers could drive more ports.

6502 INTERFACING FOR BEGINNERS: AN ASCII KEYBOARD INPUT PORT

Marvin L. De Jong
Dept. of Math-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

Introduction

Many computer systems utilize a keyboard as an input device to get data or instructions from the outside world. KIM and TIM systems interface with teletype keyboards in which a 7-bit ASCII word is sent one bit at a time to the computer. This is called "serial input" and it is very common. Of course, the computer is capable of reading all 7 bits of an ASCII word in one byte. When operated in this way the keyboard input is just another location in memory, and the mode is sometimes referred to as "parallel." We will assume that the ASCII keyboard makes all 7 bits available at once and that it produces a positive strobe signal when the ASCII data is stable.

The following ingredients are necessary to implement a parallel keyboard input port.

1) A device select pulse $\overline{DS}$ for the memory location of the keyboard
2) Three-state buffer/driver connecting the keyboard to the data bus when the device select pulse occurs, but disabling it otherwise
3) A means for the keyboard to communicate with the computer; that is, the keyboard must inform the computer that a key has been depressed
4) A means to store the data until the computer reads it into the accumulator

Previous columns have dealt with the generation of $\overline{DS}$ pulse; it will be assumed that the appropriate circuitry is available. A single Intel 8212 Eight-Bit I/O Port will be used as ingredients 2), 3), and 4) above.

The 8212 I/O Port

A logic diagram for the 8212 is shown in Figure 1. The chip contains three subsystems; the control logic (including the $\overline{DS1}$, DS2, MD, STB, $\overline{CLR}$ inputs and the $\overline{INT}$ output), the data latch, and the three-state buffers. It all looks confusing but the situation can be simplified quickly. $\overline{CLR}$ will be tied to logic 1 to disable it. MD (for mode) is tied to logic O in the input mode. Examine the AND-OR control logic carefully to see that this last step in effect connects the strobe (STB) to the C inputs of the 8-bit data latch. The keyboard strobe will be connected to STB. When the STB is at logic 1 the Q outputs of the data latch follow the DI(1-7) inputs from the keyboard. The data is latched (stored at the Q outputs) on the trailing edge of the strobe. A single key depression results in the ASCII data being stored in the 8212, with one bit left over.

Note that the STB is also connected to the C input on the service request flip-flop. The trailing edge of the strobe latches a logic O into the Q output of the flip-flop because the D input is tied to logic O. The Q output is inverted, ORed, and inverted again to produce a logic O signal at $\overline{INT}$ whenever the strobe pulse occurs. The INT signal is used to communicate with the computer, telling it that data is available. Clearly it could be connected to the interrupt (IRQ or NMI) line on the 6502 to cause an interrupt. The interrupt vector would point to a routine to read the keyboard, and would have to include a LDA KYBD instruction.

The address of KYBD appears on the address bus during the third cycle of the LDA KYBD instruction. The address bus is decoded to produce a device select pulse $\overline{DS}$ for this address, and the device select goes to pin $\overline{DS1}$ on the 8212. At the same time DS2 is brought to logic 1 by the R/W line from the 6502. When $\overline{DS1}$ is low and DS2 is high the three-state buffers are enabled and the data from the keyboard is placed on the data bus to be read into the accumulator.

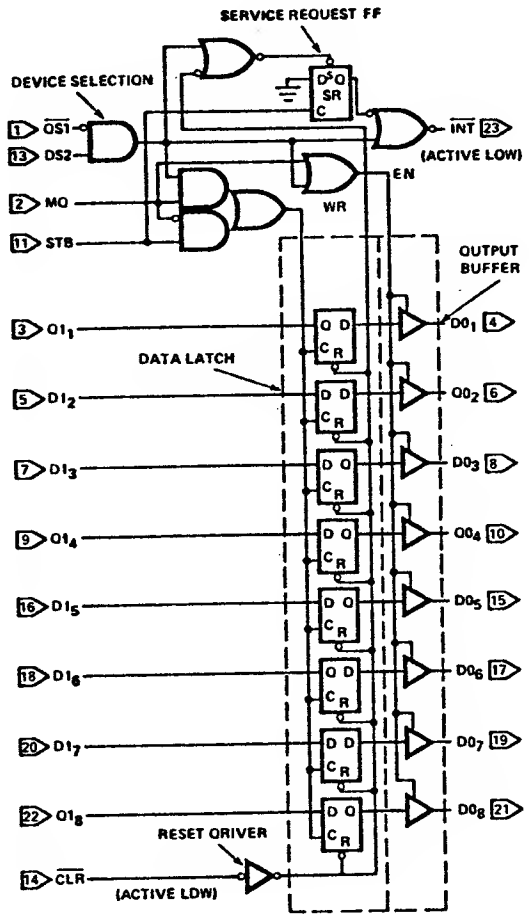


Figure 1
Logic diagram of the 8212 I/O Port.

Also observe that the $\overline{DS1}$.DS2 signal is connected to the "set" input on the service request flip-flop. This puts a logic 1 at the Q output which removes the interrupt request. The data has now been read, the interrupt cleared, and the computer is free to go on its way until another key is depressed and the entire process starts over.

Experiment with the 8212

A circuit to experiment with the 8212 is shown in Figure 2. You do not need an ASCII keyboard to construct this input port. The 74121 produces the necessary strobe signal. The data switches shown in Figure 2 can be jumper wires. For a device select I simply used the K1 select from the KIM-1, with a pull-up resistor added since the KIM-1 does not provide pull-ups for these selects. Any address decoding scheme to get a device select will do.

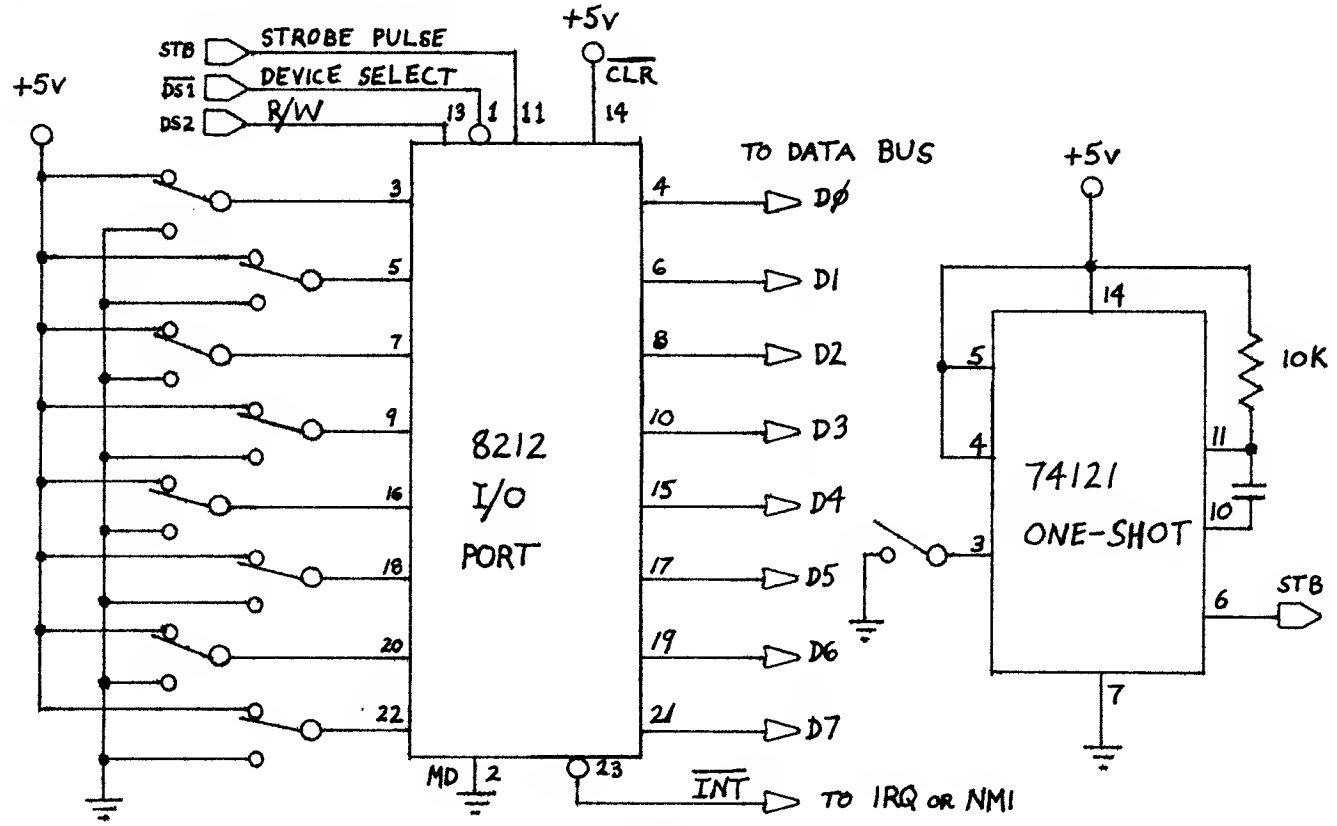


Figure 2.
8-Bit Input Port. The 74121 may be used to strobe the switch settings into the 8212. The power connections to the 8212 are pin 24 = +5V, pin 12 = GND.

Connect the data output pins to the data bus of the 6502, but leave the $\overline{INT}$ disconnected. Connect the strobe output of the 74121 to the STB pin on the 8212. Write a short program to read the 8212 and display the results on some output device. I used the following program for the KIM-1.

```
0000 AD 00 04  BEGIN  LDA   KYBD     K1 SELECT ON KIM USED
0003 85 FB            STAZ  DISP     PUT IN DISPLAY CELL
0005 20 1F 1F         JSR   SCANDS   JUMP TO KIM MONITOR
0008 4C 00 00         JMP   BEGIN    REPEAT
```

Load the program and run it. Set the switch settings for the data input to the 8212 to some value. Note that the switch settings have no effect on the displayed value. Now initiate the strobe pulse by closing the switch to the one-shot. This clocks the data into the 8212 and the computer will read it. Change the switch settings and initiate another strobe pulse. The data displayed should correspond to the switch settings. To initiate a strobe pulse the switch to the one-shot must first be opened, then closed.

Now connect the $\overline{INT}$ to the $\overline{IRQ}$ on your 6502. Run the following program:

```
0200 A2 00     BEGIN  LDXIM $00    SET UP X AS COUNTER
0202 4C 02 02  HERE   JMP   HERE   WAIT FOR INTERRUPT

0000 AD 00 04  INT    LDA   KYBD   GET DATA FROM KYBD
0003 85 10            STAZ  MEM1   SAVE DATA
0005 E8               INX          BUMP COUNTER
0006 86 11            STXZ  MEM2   SAVE COUNTER
0008 40               RTI          RETURN FROM INTERRUPT
```

Be sure to set your interrupt vector to 0000, 17FE and 17FF on the KIM-1. Run the program starting at 0200. This is just an infinite loop which initializes the X register to zero. Now hit the strobe switch. Stop the program and examine the contents of 0010. It should be identical to the switch settings for the 8212 inputs. Examine 0011 where the X register was stored. Why doesn't it read 01 corresponding to the single interrupt we produced? Because the mechanical switch used to initiate the strobe pulse was not "debounced."

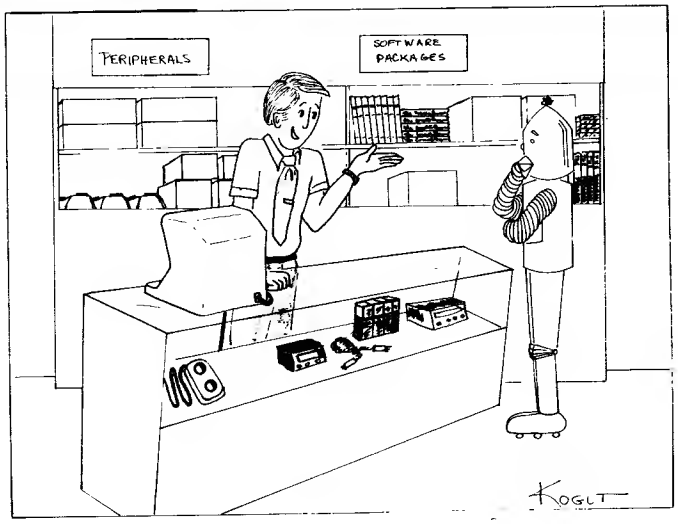
The program is very simple. The computer loops forever in the JMP HERE loop unless an interrupt occurs ($\overline{IRQ}$ pulled low by $\overline{INT}$). When the interrupt occurs the computer jumps to the interrupt routine which reads the 8212 and stores the result in 0010. X is also incremented and stored in 0011. This was done just to give you a feeling for keybounce. The program then returns to the infinite loop where vou found it when you stopped the program. Change the switch settings on the 8212 then try the program again.

Disconnect the $\overline{INT}$ from the 6502 and connect it to the DI(8) input (pin 22) on the 8212. We will now **poll** the input port to see if any data is ready. If a strobe pulse has occurred, then bit seven will be low because $\overline{INT}$ is connected to this bit. Once the 8212 is read, $\overline{INT}$ goes high as does bit seven. Here is a program to demonstrate polled service.

```
0200 20 20 02  MAIN   JSR   INPUT  SIMULATES "MAIN PROGRAM"
0203 4C 00 02         JMP   MAIN

0220                  ORG   $0220

0220 20 1F 1F  INPUT  JSR   SCANDS DISPLAY LAST INPUT DATA
0223 2C 00 04         BIT   KYBD   TEST BIT 7
0226 30 F8            BMI   INPUT  LOOP IF BIT 7 = 1
0228 AD 00 04         LDA   KYBD   ELSE, GET NEW DATA
022B 85 FB            STA   DISP   STORE IT
022D 60               RTS          RETURN TO MAIN PROGRAM
```

Play around with it changing switch settings and strobing data. Basically what it does is test bit-7 to see if any new data is available. MAIN is just a dummy program. It represents almost any program which uses a keyboard input. For example, my Micro-ADE assembler, disassembler, editor polls the keyboard for new data and my BASIC interpreter does the same thing. Both programs jump to subroutines which wait until new data has been entered from the keyboard, then return to the main program to process that information. I used JSR SCANDS in my INPUT subroutine so you could see the data on the KIM-1 display. Normally one would not use the KIM-1 display in an input routine. Rather he would "echo" the input with an output routine which would write the data on his CRT or teletype.

If you have an ASCII keyboard with a positive strobe you can do all of these same experiments but with an actual keyboard input.



May I show you something in a Ready to Ware?

by: Bertha B. Kogut

REAL-TIME GAMES ON OSI

David Morganstein
9523 48th Place
College Park, MD 20704

This note discusses how real-time games can be written for OSI Challenger systems which use a serial terminal run from the ACIA. The terminal in my system is an ADM-3A, but the same principal applies to any other. The sample program which is included does use the cursor control procedure of the ADM-3A, but it is a common enough terminal that many readers will be able to use it directly. The cursor control is accomplished in a one-line subroutine and can be changed to another procedure easily.

My original goal was to write video games, but I did not have a separate TV monitor, 440 video board and A/D convertor to do this. Fortunately, there was a way!! First, I'll discuss a procedure for polling the serial terminal keyboard and then the video display on the terminal.

The basic idea was to use a PEEK command rather than an IMPUT statement. That way the program does not have to stop while the player ponders his response. This was the ONLY way to play Lunar Lander. The typical version gives the Captain unlimited time to ponder his response and minimizes crash landings. Several articles in BYTE and elsewhere talk about using A/D convertors and joysticks. Of course, this is a fine way to go, but the same effect can be created without the added hardware.

The input byte from the ACIA appears at $EC01. To get a little appreciation for this, look at the ROM monitor routine starting at $EE00, this is called INCH in the OSD documentation. (See Figure 1.) By peeking at 64513 ($EC01), you can read the byte sent by the terminal. The only problem with this is the parity bit. That is, the bytes indicating the numbers 0-9 do not increase smoothly but have bit 7 set or not to insure parity. You can solve this by subtracting 128 when the PEEK (64513) is greater than 128. In the INCH routine this is accomplished with an AND #$7E, masking bit 7. In this way, you get values from 48 to 57 for the keys 0-9. Now these values can be used to change the burn rate of the lunar lander.

The program is fairly short and is generally self-explanatory. The polling is done in subroutine 5000. The test for 13 is needed since this is a null byte appearing before any keyboard entry has been made. As it now runs, extra boost can be given by typing a non-numeric. This should probably be prevented since it will allow a "sinking ship" to be saved, most unsporting!!

The other interesting feature is the cursor control. This is accomplished in line 6000. The ADM-3A requires two control bytes be sent, CHR$(27) and CHR$(61), in order to set up the X and Y coordinates which follow. As given in the subroutine, the X value can be from 1 to 80 and the Y from 1 to 24, which correspond to the column and row (counting from the top left) of the position to be printed. Be careful when using this to not exceed these ranges. The cursor control is used to set-up a "lander control panel" and then update the "meter readings" as the play progresses.

If your wondering what line 500 does, its used for timing. By adjusting the variable DE(lay), the speed of the game can be changed slightly. I was shooting for a twice per second update on the panel. Unfortunately, when the LOW EUEL WARNING comes on the timing changes. Well, you can't have everything. (I'm sure somebody out there will figure out how to correct this....)

```
FE00 AD 00 FC  START  LDA   $FC00
FE03 4A               LSRA
FE04 90 FA            BCC   START
FE06 AD 01 FC         LDA   $FC01
FE09 29 7F            ANDIM $7F
FE0B 48               PHA
FE0C AD 00 FC         LDA   $FC00
FE0F 4A               LSRA
FE10 4A               LSRA
FE11 90 F9            BCC   $FE0C
FE13 68               PLA
FE14 8D 01 FC         STA   $FC01
FE17 60               RTS
FE18 20 00 FE         JSR   START
FE1B C9 52            CMPIM $52
FE1D F0 16            BEQ   $FE35
FE1F C9 30            CMPIM $30
FE21 30 F5            BMI   $FE18
FE23 C9 3A            CMPIM $3A
FE25 30 0B            BMI   $FE32
FE27 C9 41            CMPIM $41
FE29 30 ED            BMI   $FE18
FE2B C9 47            CMPIM $47
```

```
100 PRINTCHR$(26):X=25:Y=10:GOSUB6000
104 PRINT"L U N A R   L A N D E R ":Y=12:GOSUB6000
106 INPUT"DO YOU NEED INSTRUCTIONS (Y/N) ";N$
110 IFN$="N"GOTO190
115 PRINT:PRINT
120 PRINTTAB(10)"THIS IS A REAL TIME LUNAR LANDER SIMULATION.
130 PRINTTAB(10)"TO PLAY, MERELY ENTER THE POUNDS OF
140 PRINTTAB(10)"FUEL WHICH YOU WISH TO BURN BY TYPING A DIGIT (0-9).
150 PRINTTAB(10)"THE NINE GIVES MAXIMUM BURN, SLOWING YOU DOWN AT THE
155 PRINTTAB(10)"FASTEST RATE.  A ZERO GIVES NO BURN AND LETS YOU FRE
160 PRINTTAB(10)"FALL.":PRINT:INPUT"   READY...TYPE GO  ";N$
190 PRINTCHR$(26):Y=4:X=28:GOSUB6000:PRINT"TIME TO FUEL EXHAUSTION"
200 X=20:Y=7:GOSUB6000:PRINT"BURN RATE"
220 X=50:GOSUB6000:PRINT"FUEL"
230 Y=8:X=20:GOSUB6000:PRINT(LBS/SEC)"X=50:GOSUB6000:PRINT"(LBS)"
240 Y=12:X=20:GOSUB6000:PRINT"VELOCITY":X=50:GOSUB6000:PRINT"ALTITUDE
250 Y=13:X=20:GOSUB6000:PRINT"(FT/SEC)":X=50:GOSUB6000:PRINT"  (FT)"
260 Y=18:X=20:GOSUB6000:PRINT"ESTIMATED TIME TO LANDING "
270 Y=22:X=1:GOSUB6000:FORI=1TO79:PRINT"-";:NEXTI
275 Y=23:X=1:GOSUB6000:PRINT"0  "
280 FORI=1TO7:X=10*I:GOSUB6000:PRINTI;:NEXTI
290 X=30:Y=24:GOSUB6000:PRINT"ALTITUDE (X10,000 FT.)":GOSUB6000
310 VE=-100:MT$="            ":FU=10000:AL=80000:DE=5:BU=32
320 FORT=1TO10000
330 IFT/2=INT(T/2)THENPRINTCHR$(7);
340 VE=VE+((BU-32)*25EB)/(25EB+AL*AL))
345 VE=INT(VE)
350 AL=AL+INT(VE/2)
360 IFAL<0GOTO3000
370 IFFU<500THENGOSUB2000
380 FU=FU-BU/2
385 IFFU<=0THENFU=0:BU=0
390 IFBU<=0THENB$="NO BURN":GOTO410
400 B$=STR$(INT(FU/BU))
410 X=38:Y=5:GOSUB6000:PRINTMT$:GOSUB6000:PRINTB$
420 X=21:Y=9:GOSUB6000:PRINTBU:X=50:GOSUB6000:PRINTFU
430 X=22:Y=14:GOSUB6000:PRINTVE:X=50:GOSUB6000:PRINTAL
440 IFVE>=0THENA$="ESCAPE":GOTO460
450 A$=STR$(INT(AL/ABS(VE)))
460 Y=19:X=38:GOSUB6000:PRINTMT$:GOSUB6000:PRINTA$
461 TA=INT((AL+500)/1000):IFTA>80THENTA=80
462 IFTA<1THENTA=1
463 Y=21:X=TA+1:GOSUB6000
465 IFFU=0GOTO500
470 GOSUB5000:IFZ=13GOTO500
480 BU=12+4*(Z-48)
490 IFZ=48THENBU=0
500 FORTI=1TODE:A=SIN(10):NEXTTI
505 VP=VE:AP=AL
510 NEXTT
2000 FORJ=1TO2
2005 X=36:Y=12:GOSUB6000:PRINT"LOW FUEL"
2010 Y=13:GOSUB6000:PRINT"WARNING"
2020 A=SIN(10)
2030 GOSUB6000:PRINTMT$:Y=12:GOSUB6000:PRINTMT$
2035 A=SIN(10)
2040 NEXTJ
2050 DE=I:RETURN
3000 SP=(VP+VE)/2
3010 IFSP<-25GOTO3200
3015 PRINT:PRINT
3020 PRINTTAB(20)"CONGRATULATIONS, YOU TOUCHED DOWN AT A MERE "
3030 PRINTTAB(30)SP;" FT./SEC.  A SAFE LANDING !!!"
3040 PRINT:PRINTTAB(20)" DO YOU WANT TC TRY AGAIN AND"
3050 PRINTTAB(20)" ";:INPUT"PROVE IT WASN'T LUCK ";N$
3060 IFN$="N"THENRUN"BEXEC*"
3070 GOTO190
3200 PRINTCHR$(26)
3210 N=40
3220 FORI=1TON:X=1+INT(79*RND(I)):Y=1+INT(23*RND(1))
3225 GOSUB6000:PRINTCHR$(33+INT(15*RND(1))):GOSUB6000:NEXTI
3230 X=20:Y=10:GOSUB6000:PRINT"YOU JUST BLEW A CRATER,"
3240 Y=11:GOSUB6000:PRINTABS(VE);" FEET IN DIAMETER, ON THE
3250 Y=12:GOSUB6000:PRINT"SURFACE OF THE MOON.  BETTER TRY AGAIN...
3260 Y=14:GOSUB6000:INPUT" READY (Y/N) ";N$
3270 GOTO190
5000 Z=PEEK(64513)
5005 IFZ=13THEN RETURN
5010 IFZ>128THENZ=Z-128:RETURN
6000 PRINTCHR$(27);CHR$(61);CHR$(Y+31);CHR$(X+31);:RETURN
```

650X OPCODE SEQUENCE MATCHER

J. S. Green
807 Bridge Street
Bethlehem, PA 18018

The motivation for writing this program stemmed from the fact that I have two machine code versions of the same 650X assembler (ASM65 by Wayne Wall, dated 1 May 77 and 13 Jun 77 respectively) but I only have a listing of the older version. Both are just short of 4 K bytes long. I wished to make some local changes to the newer version and therefore needed to establish a means of correspondence between it and the listing. A disassembler is helpful here but not adequate because of discontinuities in the two codes which make forward references very difficult to correlate manually.

I felt that when a program has been heavily modified, many opcode sequence segments whould remain constant even while their respective operands differ. Therefore, what was needed was a program that would correlate and point to parallel sequences of opcodes.

Several assumptions were made in order to simplify the programming task. It was presumed that the basic order of appearance of major portions of the code would be the same since there seemed to be little advantage in shuffling the deck, as it were. Also, in order to minimize the effect of spurious matches, it was decided that only significant sequences need be reported and that no portion of the code would be reported as a match more than once. This position saves the program, for example, from reporting every possible LDA,STA opcode sequence pairing (or even all of those of the same address mode).

Process Description

As written, the scanning process of the matching program starts at the beginning of the two code strings, A and B, to be examined. Both initial positions are assumed to contian opcodes. An index or pointer to the B string is, in effect, moved along B, from opcode to opcode, until a match with the current A string opcode is found. If no match is found before the B list is exhausted, the A pointer is moved to the next A opcode position while the B pointer is reset to its previous starting point. This general procedure is repeated until the A list is exhausted, at which time the program terminates.

When a match is found, both pointers are moved together along their respective lists, from opcode to opcode, until the opcodes fail to match each other. If the matching sequence is significantly long the size and the start and end of both segments is displayed. The search for additional matching segments is resumed from the end of the just-reported segments so that their opcode elements cannot be matched more than once.

If the completed sequence is not significant, it is not displayed and the search is resumed from where the short sequence began, as if there had been no match at all.

The definition of significance refers to the minimum acceptable number of matching codes in a continuous sequence. The particular values used are left to the user. While our experience has shown a minimum value of eight to be useful, the actual values should reflect the length of the code being examined and the degree to which it has been hacked up.

The effect of a too-low significance value often results in a fewer number of matches being reported, rather than more as one might expect. This is because a spurious match of short segments can have the effect of masking out longer possible matches which would use the same code items were they still available.

Operation

To operate the opcode matching program both lists of code must be in memory. They may be in ROM. They need not be at their operating address. (Indeed, if they have the same address at least one must be somewhere else anyway). Since the matching program reports storage, rather than operating addresses it is useful to choose storage addresses that have some degree of correspondence to the operating addresses, e.g., code operating at $21E3 might be stored at $41E3.

Enter initial values (all in hex LO,HI) as follows:

$0000,01	Significance value
$0002,03	Start of list A
$0004,05	Start of list B
$0006,07	End of list A
$0008,09	End of list B

Only the starting address will be modified during program execution. The program will initially assume that the value at the start location is an opcode.

To run the program enter at OPMACH. As written, it will terminate by jumping to the monitor from END01. The routine may be made into a subroutine by placing an RTS here.

Since the program cranks the data a lot, there will be what seem to be long pauses between outputs. The program requires about 2 minutes to compare the aforementioned assemblers.

Results

Several sets of results, using significance values of $06, $08 and $0A are shown below. In order to have both versions of code resident at the same time, it was necessary to store one version, at address $4000.

About 64 percent of the code of the two versions of the assembler correlate when a significance value of 8 is used. This is a reasonable percentage when one considers the fact that the non-significant, non-reported, sequences are easily identified since they lie in the same relative position between reported sequences.

An extensive manual comparison of the two code sets was made. (So much for the work-saving aspects of the program!) No false matches were identified when a significance value of 8 was used.

Variations for Text Processing

Interesting variants of the program are possible. By altering or replacing the list pointer increment routines, AINC and BINC, the nature of the list pointer incrementation may be changed from the current conditional increment based on opcode to some other condition or to a constant such as plus one.

With a constant increment of one, the matching program may be used to compare sequences of any

textural material in a somewhat crude, one for one fashion.

By having separate increment subroutines when seeking to locate the start of a matching segment in contrast to the incremental routines used when "running-out" a sequence, some fairly powerful text processing capabilities may be obtained at little additional cost. For example, when seeking to locate matching segments in natural language text, we might wish to start with the initial character of alphabetic strings, i.e., words. Therefore, by incrementing past all non-alphabetic characters to the next alphabetic character we can both speed up the process and insure that our sequences start with (what we have operationally defined as) words.

Similar techniques may be employed in the (now separate) within sequence increment routines to ingore, (i.e., increment past,) any non-alphabetic characters such as control characters, numbers, punctuation or whatever we like. Thus we are able to obtain a far more flexible and hopefully more useful definition of a matching sequence.

Conclusions

The general techniques illustrated here are both effective and useful. The conditional matching approach has not been fully explored, but it is clear that it has interesting possibilities in the area of text processing. In the present application, correlating two lenghty strings of machine code, the approach made practical what otherwise would have been a difficult and dull task.

```
                        ;       **** OPCODE SEQUENCE MATCHER ****
                        ;       VERSION 1.04. 18 AUG 78
                        ;
                        ;       COPYRIGHT,1978
                        ;       COMMERCIAL RIGHTS RESERVED
                        ;       EXCEPT AS NOTED BY

                        ;       J. S. GREEN. COMPUTER SYSTEMS
                        ;       807 BRIDGE STREET
                        ;       BETHLEHEM. PA  18018
                        ;       (215) 867-0924
                        ;
                        ;       NOTE: THE BYTCNT SUBROUTINE IS FROM
                        ;       H. T. GORDON IN DDJ. #22 P.5.
                        ;       (COPYRIGHT BY PEOPLE'S COMPUTER COMPANY)

                                .LOC  $0000
                        ;
                        ;       USER DEFINED VARIABLES (LO,HI)
0000 00 00      SIGNIF: .WORD             ;SIGNIFICANCE
0002 00 00      ABASE:  .WORD             ;START OF LIST A
0004 00 00      BBASE:  .WORD             ;START OF LIST B
0006 00 00      AMAX:   .WORD             ;END OF LIST A
0008 00 00      BMAX:   .WORD             ;END OF LIST B
                        ;
                        ;       OTHER PROGRAM VARIABLES
000A 00 00      APOINT: .WORD             ;LIST A POINTER
000C 00 00      BPOINT: .WORD             ;LIST B POINTER
000E 00 00      ASAVE:  .WORD             ;LIST A SEQUENCE START
0010 00 00      BSAVE:  .WORD             ;LIST B SEQUENCE START
0012 00 00      COUNT:  .WORD             ;SEQUENCE COUNTER
                        ;
                        ;       EXTERNAL SUBROUTINES (IN KIM)
                                .DEF  START=$1C4F         ;MONITOR RETURN POINT
                                .DEF  CRLF=$1E2F          ;CARRIAGE RETURN
                                .DEF  OUTCH=$1EA0         ;DISPLA A CHAR
                                .DEF  PRTBYT=$1E3B        ;DISPLA HEX BYTE
                                .DEF  OUTSP=$1E9E         ;DISPLA A SPACE
                        ;
                                .LOC  $0200
                        ;
0200 20 2F 1E   OPMACH: JSR   CRLF
0203 A2 29              LDX#  $29         ;SIGN + HEADER COUNT
0205 BD 4F 03   OPMCH1: LDAX  SIGN        ;DISPLAY HEADER
0208 20 A0 1E           JSR   OUTCH
020B CA                 DEX
020C 10 F7              BPL   OPMCH1
020E A5 01              LDA   SIGNIF+1
0210 20 3B 1E           JSR   PRTBYT      ;DISPLAY SIGNIF HI
0213 A5 00              LDA   SIGNIF
0215 20 3B 1E           JSR   PRTBYT      ;DISPLAY SIGNIF LO
0218 20 2F 1E           JSR   CRLF
021B 20 3B 03           JSR   BASPNT      ;POINTERS=BASES
```

```
021E A5 03      DO1:    LDA   ABASE+1
0220 C5 07              CMP   AMAX+1
0222 30 09              BMI   IF1         ;BR IF WHOLE JOB NOT DONE
0224 A5 02              LDA   ABASE
0226 C5 06              CMP   AMAX
0228 30 03              BMI   IF1         ;BR IF WHOLE JOB NOT DONE
022A 4C B7 02           JMP   ENDO1       ;HERE IF WHOLE JOB DONE
022D A2 00      IF1:    LDX#  0           ;DOES CURRENT PAIR MATCH•
022F A1 0A              LDAX@ APOINT
0231 C1 0C              CMPX@ BPOINT
0233 D0 64              BNE   ELS1        ;BR IF NOT THE SAME
0235 86 12      THEN1:  STX   COUNT       ;HERE ON SAME
0237 86 13              STX   COUNT+1     ;CLEAR THE COUNTER
0239 A2 03              LDX#  3
023B B5 0A      THN1A:  LDAX  APOINT      ;SAVES=POINTERS
023D 95 0E              STAX  ASAVE
023F CA                 DEX
0240 10 F9              BPL   THN1A
0242 A2 00      DO2:    LDX#  0           ;DO TILL NOT THE SAME
0244 A1 0A              LDAX@ APOINT
0246 C1 0C              CMPX@ BPOINT
0248 D0 26              BNE   ENDO2       ;BR IF NOT THE SAME
024A A5 0B              LDA   APOINT+1
024C C5 07              CMP   AMAX+1
024E 30 06              BMI   EXP21       ;BR IF LESS THAN
0250 A5 0A              LDA   APOINT
0252 C5 06              CMP   AMAX
0254 10 1A              BPL   ENDO2       ;BR TO ENDO
0256 A5 0D      EXP21:  LDA   BPOINT+1
0258 C5 09              CMP   BMAX+1
025A 30 06              BMI   EXP22       ;BR IF LESS THAN
025C A5 0C              LDA   BPOINT
025E C5 08              CMP   BMAX
0260 10 0E              BPL   ENDO2       ;BR TO ENDO IF LIMIT REACHED
0262 20 BA 02   EXP22:  JSR   AINC        ;MOVE A POINTER TO NEXT A OPCODE
0265 20 CE 02           JSR   BINC        ;MOVE B POINTER TO NEXT B OPCODE
0268 E6 12              INC   COUNT
026A D0 D6              BNE   DO2
026C E6 13              INC   COUNT+1
026E D0 D2              BNE   DO2         ;BR ALWAYS TO TOP OF DO
0270 EA         ENDO2:  NOP               ;A WASTED BYTE FOR "STRUCTURE"
0271 A5 13      IF2:    LDA   COUNT+1
0273 C5 01              CMP   SIGNIF+1
0275 30 0F              BMI   ELS2        ;BR IF NOT SIGNIF
0277 A5 12              LDA   COUNT
0279 C5 00              CMP   SIGNIF
027B 30 09              BMI   ELS2
027D 20 FE 02   THEN2:  JSR   REPORT      ;HERE ON SIGNIF. OUTPUT RESULT
0280 20 45 03           JSR   PNTBAS      ;POINTERS=BASES
0283 4C 96 02           JMP   ENDIF2
0286 A2 01      ELS2:   LDX#  1
0288 20 3D 03           JSR   BASPT1      ;APOINT=ABASE
028B A5 10              LDA   BSAVE
028D 85 0C              STA   BPOINT
028F A5 11              LDA   BSAVE+1
0291 85 0D              STA   BPOINT+1
0293 20 CE 02           JSR   BINC
0296 4C 9C 02   ENDIF2: JMP   ENDIF1
0299 20 CE 02   ELS1:   JSR   BINC        ;
029C EA         ENDIF1: NOP               ;ANOTHER SOP TO "STRUCTURE"
029D A5 0D      IF3:    LDA   BPOINT+1
029F C5 09              CMP   BMAX+1
02A1 30 11              BMI   ENDIF3      ;BR IF NOT DONE
02A3 A5 0C              LDA   BPOINT
02A5 C5 08              CMP   BMAX
02A7 30 0B              BMI   ENDIF3      ;BR IF NOT DONE
02A9 20 3B 03   THEN3:  JSR   BASPNT
02AC 20 BA 02           JSR   AINC
02AF A2 01              LDX#  1
02B1 20 47 03           JSR   PNTBS1
02B4 4C 1E 02   ENDIF3: JMP   DO1
02B7 4C 4F 1C   ENDO1:  JMP   START
```

```
                        ;
                        ;       SUBROUTINES FOLLOW
                        ;
                        ;       MOVE TO NEXT A OPCODE
02BA A2 00      AINC:   LDX#  0
02BC A1 0A              LDAX@ APOINT      ;GET OPCODE
02BE 20 E2 02           JSR   BYTCNT      ;CALCULATE SIZE
02C1 8A                 TXA               ;RESULT RETURNED IN X
02C2 18                 CLC
02C3 65 0A              ADC   APOINT      ;ADD RESULT TO POINTER
02C5 85 0A              STA   APOINT
02C7 A5 0B              LDA   APOINT+1
02C9 69 00              ADC#  0
02CB 85 0B              STA   APOINT+1
02CD 60                 RTS
                        ;
                        ;       MOVE TO NEXT B OPCODE
02CE A2 00      BINC:   LDX#  0
02D0 A1 0C              LDAX@ BPOINT      ;GET OPCODE
02D2 20 E2 02           JSR   BYTCNT      ;CALCULATE SIZE
02D5 8A                 TXA               ;RESULT RETURNED IN X
02D6 18                 CLC
02D7 65 0C              ADC   BPOINT      ;ADD RESULT TO POINTER
02D9 85 0C              STA   BPOINT
02DB A5 0D              LDA   BPOINT+1
02DD 69 00              ADC#  0
02DF 85 0D              STA   BPOINT+1
02E1 60                 RTS
                        ;
                        ;       CALCULATE SIZE OF OPERAND (+1)
                        ;       BY H. T. GORDON (SEE DDJ #22, P.5)
02E2 A2 01      BYTCNT: LDX#  1
02E4 2C E8 02           BIT   BYTCNT+6    ;TEST BIT 3
02E7 D0 08              BNE   HAFOP       ;ALL X(8-F)
02E9 C9 20              CMP#  $20
02EB F0 0E              BEQ   THREE       ;ONLY $20
02ED 29 9F              AND#  $9F         ;BITS 5,6 OUT
02EF D0 0B              BNE   TWO         ;ALL EXCEPT (0,4,6)0
02F1 29 15      HAFOP:  AND#  $15         ;RETAINS ONLY BITS 0,2,4
02F3 C9 01              CMP#  1
02F5 F0 05              BEQ   TWO         ;X(9,B)
02F7 29 05              AND#  5           ;BIT 4 OUT
02F9 F0 02              BEQ   ONE         ;X(8,A) AND (0,A,6)0
02FB E8         THREE:  INX               ;RESID. X(9-F)
02FC E8         TWO:    INX
02FD 60         ONE:    RTS
                        ;
                        ;       DISPLAY SIGNIFICANT SEQUENCE LIMITS
02FE A2 01      REPORT: LDX#  1
0300 B5 12      REPT1:  LDAX  COUNT       ;OUTPUT EXTENT OF MATCH
0302 20 3B 1E           JSR   PRTBYT
0305 CA                 DEX
0306 10 F8              BPL   REPT1
                ;       OUTPUT MULTIPLE SPACES
0308 20 31 03           JSR   OUTSP4      ;FOUR SPACES
030B A2 00              LDX#  0
030D B5 0F      REPT2:  LDAX  ASAVE+1     ;OUTPUT START AND
030F 20 3B 1E           JSR   PRTBYT      ; END ADDR OF
0312 B5 0E              LDAX  ASAVE       ; BOTH SEGMENTS
0314 20 3B 1E           JSR   PRTBYT
0317 20 34 03           JSR   OUTSP2
031A B5 0B              LDAX  APOINT+1
031C 20 3B 1E           JSR   PRTBYT
031F B5 0A              LDAX  APOINT
0321 20 3B 1E           JSR   PRTBYT
0324 20 31 03           JSR   OUTSP4
0327 E8                 INX
0328 E8                 INX
0329 E0 03              CPX#  3
032B 30 E0              BMI   REPT2
032D 20 2F 1E           JSR   CRLF
0330 60                 RTS
```

```
                        ;
0331 20 34 03   OUTSP4: JSR   OUTSP2      ;4 SPACES
0334 20 9E 1E   OUTSP2  JSR   OUTSP       ;2 SPACES
0337 20 9E 1E           JSR   OUTSP
033A 60                 RTS
                        ;
                        ;       MOVE ABASE & BBASE TO APOINT & BPOINT
033B A2 03      BASPNT: LDX#  3
033D B5 02      BASPT1  LDAX  ABASE
033F 95 0A              STAX  APOINT
0341 CA                 DEX
0342 10 F9              BPL   BASPT1
0344 60                 RTS
                        ;
                        ;       MOVE APOINT & BPOINT TO ABASE & BBASE
0345 A2 03      PNTBAS: LDX#  3
0347 B5 0A      PNTBS1: LDAX  APOINT
0349 95 02              STAX  ABASE
034B CA                 DEX
034C 10 F9              BPL   PNTBS1
034E 60                 RTS
                        ;
                        SIGN:   .ASCII          ' = FINGIS  '
034F 20
0350 3D
0351 20
0352 46
0353 49
0354 4E
0355 47
0356 49
0357 53
0358 20
0359 20

                        HEADER: .ASCII          'OT   MORF    OT   MORF    EZIS
035A 4F
035B 54
035C 20
035D 20
035E 20
035F 4D
0360 4F
0361 52
0362 46
0363 20
0364 20
0365 20
0366 20
0367 20
0368 4F
0369 54
036A 20
036B 20
036C 20
036D 4D
036E 4F
036F 52
0370 46
0371 20
0372 20
0373 20
0374 20
0375 45
0376 5A
0377 49
0378 53

                        ;
                                .END
```

```
0379 .
0000 SIGNIF     02BA AINC
0002 ABASE      02CE BINC
0004 BBASE      0271 IF2
0006 AMAX       0286 ELS2
0008 BMAX       02BA AINC
000A APOINT     02CE BINC
000C BPOINT     0271 IF2
000E ASAVE      0286 ELS2
0010 BSAVE      027D THEN2
0012 COUNT      02FE REPORT
1C4F START      0345 PNTBAS
1E2F CRLF       0296 ENDIF2
1EA0 OUTCH      033D BASPT1
1E3B PRTBYT     029C ENDIF1
1E9E OUTSP      029D IF3
0200 OPMACH     02B4 ENDIF3
0205 OPMCH1     02A9 THEN3
034F SIGN       0347 PNTBS1
033B BASPNT     02E2 BYTCNT
021E DO1        02F1 HAFOP
022D IF1        02FB THREE
02B7 ENDO1      02FC TWO
0299 ELS1       02FD ONE
0235 THEN1      0300 REPT1
023B THN1A      0331 OUTSP4
0242 DO2        030D REPT2
0270 ENDO2      0334 OUTSP2
0256 EXP21      035A HEADER
0262 EXP22
```

	SIZE	FROM	TO	FROM	TO	SIGNIF = 0006
	0026	2000	2052	4000	4052	
x	0007	2069	207B	4093	40A5	
x	0006	2099	20A5	42C2	42CE	
x	0006	2224	2234	437C	438C	
x	000A	2237	224D	4784	479A	
x	000B	274E	2761	479D	47B0	
x	0008	279D	27AC	47BB	47CA	
	007A	28D1	29BE	47CF	48BC	
	0008	29BF	29D1	48BC	48CE	
	0019	29DB	2A0D	48CE	4900	
	004D	2A17	2AC6	492D	49DC	
	002E	2ACB	2B33	49E1	4A49	
	0035	2B6E	2BE5	4A49	4AC0	
	000C	2BF2	2C04	4ACD	4ADF	
	0106	2CE2	2F01	4B27	4D46	

Note:
items tagged with an 'x' represent false matches.

SIZE	FROM	TO	FROM	TO	SIGNIF = 0008
0026	2000	2052	4000	4052	
003D	206C	20F0	4052	40D6	
0020	20F3	213C	40D6	411F	
001F	213C	2180	4122	4166	
000E	2187	21A7	416D	418D	
0046	21AA	224D	4198	423B	
0087	2275	2394	4258	4377	
0009	23A8	23BB	438F	43A2	
0126	23C0	25E6	43A2	45C8	
004C	25F1	269F	45C8	4676	
0087	26C1	27C1	4692	4792	
000E	27C8	27E2	479D	47B7	
000C	27E5	27F9	47BB	47CF	
007A	28D1	29BE	47CF	48BC	
0008	29BF	29D1	48BC	48CE	
0019	29DB	2A0D	48CE	4900	
004D	2A17	2AC6	492D	49DC	
002E	2ACB	2B33	49E1	4A49	
0035	2B6E	2BE5	4A49	4AC0	
000C	2BF2	2C04	4ACD	4ADF	
0087	2DE5	2F01	4C2A	4D46	

SIZE	FROM	TO.	FROM	TO	SIGNIF = 000A
0026	2000	2052	4000	4052	
003D	206C	20F0	4052	40D6	
0020	20F3	213C	40D6	411F	
001F	213C	2180	4122	4166	
000E	2187	21A7	416D	418D	
0046	21AA	224D	4198	423B	
0089	2271	2394	4254	4377	
0126	23C0	25E6	43A2	45C8	
004C	25F1	269F	45C8	4676	
0089	26BC	27C1	468D	4792	
000E	27C8	27E2	479D	47B7	
000C	27E5	27F9	47BB	47CF	
007A	28D1	29BE	47CF	48BC	
001D	29D1	2A0D	48C4	4900	
004D	2A17	2AC6	492D	49DC	
002E	2ACB	2B33	49E1	4A49	
0035	2B6E	2BE5	4A49	4AC0	
000C	2BF2	2C04	4ACD	4ADF	
0089	2DE1	2F01	4C26	4D46	

CASSETTE TAPE CONTROLLER

Fred Miller
7 Templar Way
Parsippany, NJ 07054

The ideal tape storage facility for micro-systems would be one in which the micro has complete control of all tape movement and play/record functions without "operator intervention" e.g. pushing buttons. Unfortunately most of us have budgets which only allow use of lower cost audio cassette units. Short of massive mechanical rebuilding, these units can only be externally controlled with a motor on/off function after the "operator" has set the proper record/play keys. All too often we goof and press the wrong button, have to move cassettes from one unit to another, or simply forget to set up the units at the right time.

The Cassette Tape Controller (CTC) described below offers a reasonably inexpensive capability as a compromise in the provision of automatic tape control for a KIM-1 system. CTC is a combination of a seven-IC hardware board and supporting software routines. It was developed to control two Pioneer Centrex KD-12 cassette units. The concept could be extended to more than two units or perhaps other models.

A summary of the functions provided are:

(1) Provide software-driven capability to start and stop a specific tape recorder by opening/closing the "remote control" circuit of the recorder (normally controlled by a switch on an external microphone).

(2) Provide software-driven capability to route the input (record) or output (playback) signals as appropriate.

(3) Provide override manual controls (toggles) to also accomplish (1) and (2), above.

(4) Light panel indicators (LEDs) associated with the play or record functions selected for each cassette unit as set by software or manual controls.

(5) Sense whether the selected tape recorder is set to play or record, or neither.

(6) Sense the position of auxiliary toggles for setting software options, etc., (option switches.

(7) Light indicators (LEDs) associated with the auxiliary toggles for operator communications.

(8) Provide an audible "beep" under software control.

CTC General Description

The Cassette Tape Controller is a hardware/software facility to assist in the operation and use of audio cassette tape recorders for data read/write functions. The hardware provides the interface from a KIM-1 to two Pioneer Centrex KD-12 tape recorders. Besides the cassette input and output lines from KIM-1 four other lines (bit ports) are required for software control of the hardware.

The software and hardware control the recorder's motor circuits and determine if the appropriate manual keys on the recorder are set correctly. The software can provide alternative action (alert the operator or try another unit) in the case of improperly set keys.

The specific software illustrated below is written to "search" for a unit which is set in either a "read" (playback) or "write" (record) mode. If none is found in the desired mode, an audible tone is sounded and the search is continued. The visible indication of each of the "read" or "write" LEDs blinking along with the audible tone provides the operator with a quick clue as to the erroneous settings. If the appropriate tapes are "mounted" the operator simply depresses the "requested" cassette unit key. Subsequent references by the software would locate the preset unit without communicating to the operator.

Additional facilities are built into the CTC hardware/software at little extra cost. These include the separately accessible audible tone and two option toggles with accompanying panel indicator LEDs. The toggles can be used for setting options selected by the operator and tested by the software. The associated indicators can also be used for some optional communication purposes. A third switch (momentary toggle or pushbutton) is used as a "break" command for software testing. A layout of the related hardware control panel is shown in Fig. 1.

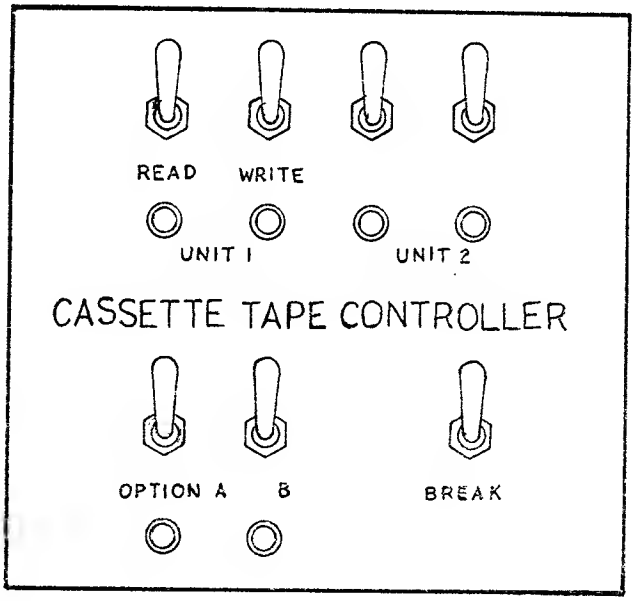


Figure 1.
Suggested Panel Layout
for Cassette Tape Controller

Hardware Description

A key to the logic of CTC is the ability to sense actual cassette unit key settings. By sensing voltage levels at two externally accessible points in the KD-12 circuitry it is possible to determine one of the following states:

(1) unit set for read (playback) or fast forward or rewind

(2) unit set for write (record)

(3) no keys depressed

The circuit shown in Fig. 2 uses two ICs to address a function, one to enable and the other to sense results of enabling. This logic is further described in the comments accompanying the software source listing. Four non-critical DPDT relays are used to allocate signals and control

motor circuits. The additional circuits, (1) pulse an audible tone generator, (2) light LED indicators, or, (3) sense toggle switch positions all depending upon addressed functions.

Three bits (PB 0-2) from KIM-1 Applications Port B are used to address the functions. Another bit line (PB 3) of the same port is used to feed status back to KIM-1.

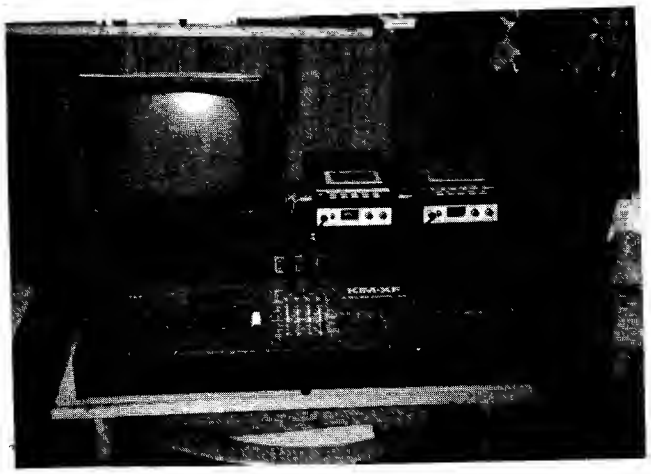
The KD-12 units are operated from external battery power (continually trickle-charged) to provide the most stable unit operation. HYPERTAPE speeds are extremely reliable in this configuration.

Software Description

The controlling software consists of a series of routines which are accessible from user programs. The software shown in Fig. 3 is designed to "seek out" a cassette unit which is set for a given function, e.g., read. A brief study of the routines will show how this can be replaced or amended to select only a given cassette unit for a specific function. The additional routines are provided for "testing" the optional toggle switches, etc. Many of the routines are useful for other than tape cassette control, e.g., a JSR to BELL provides an audible "beep".

Conclusion

The hardware and software described have been working very satisfactorily on the author's system for well over a year. The CTC software (along with tape and record I/O routines based on the HYPERTAPE routines) have been committed to EPROM (2708). Access to this capability is easy and provides convenient operation of tape file processing from user software programmed in any language used on the KIM-1 micro (BASIC, Assembler, HELP, etc.). Although the operator still must press the keys on the cassette units, the CTC system can save many a "rerun" or clobbered files due to careless operations.



Author's KIM Based System

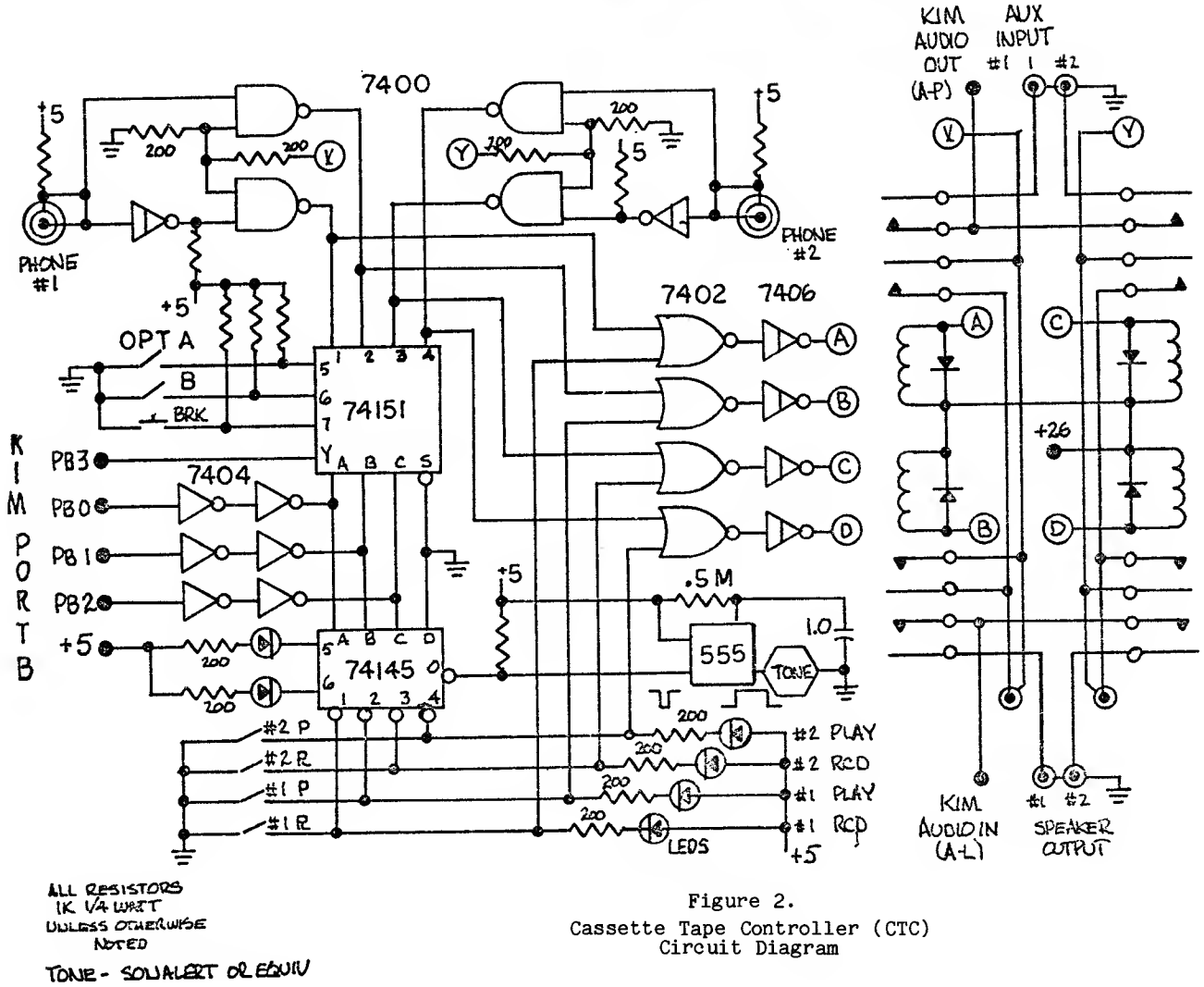


Figure 2.
Cassette Tape Controller (CTC)
Circuit Diagram

```
0010:  0200             KXFTAP ORG   $0200
0020:
0030:                  **********************
0040:                  *                    *
0050:                  *   CASSETTE TAPE    *
0060:                  * CONTROLLER (CTC)   *
0070:                  *   BY F.MILLER      *
0080:                  *                    *
0090:                  **********************
0100:
0110:                  *** KIM & ZERO PAGE PARAMETERS ***
0120:
0130:  0200             PBD    *     $1702
0140:  0200             PBDD   *     $1703
0150:  0200             TPFCT  *     $00EF
0160:  0200             INIT   *     $1E8C
ID=02

0010:                  *** TAPE CASSETTE READ ROUTINES ***
0020:
0030:  0200 D8          RDTAPE CLD
0040:  0201 A9 02              LDAIM $02    TEST FOR UNIT#1 READY
0050:  0203 20 1B 02           JSR   TPTEST FOR READ?
0060:  0206 F0 0C              BEQ   CREAD  ...YES
0070:  0208 A9 04              LDAIM $04    ...NO, UNIT#2 READY?
0080:  020A 20 1B 02           JSR   TPTEST
0090:  020D F0 05              BEQ   CREAD  ...YES
0100:  020F 20 2B 02           JSR   BELL   ...NO, SOUND SIGNAL AND
0110:  0212 D0 EC              BNE   RDTAPE TRY AGAIN.
0120:
0130:  0214 EA          CREAD  NOP
0140:                  .
0150:                  .
0160:                  . ROUTINE FOR READING TAPE
0170:                  . GOES HERE
0180:                  .
0190:                  .
0200:
0210:  0215 20 33 02           JSR   CTLOFF TURN OFF CASSETTE MOTOR
0220:  0218 4C 8C 1E  RDEXIT   JMP   INIT   AND RETURN VIA KIM INIT
ID=03

0010:                  *** CASSETTE SUPPORT RTNS ***
0020:
0030:  021B 85 EF      TPTEST STA   TPFCT  SAVE UNIT/FCT
0040:  021D 8D 02 17           STA   PBD    PORT B CONTROL DATA
0050:  0220 20 3C 02           JSR   DELAY  ALLOW RELAY SETTLE
0060:  0223 AD 02 17           LDA   PBD    CK BITS 0-3 = TO
0070:  0226 29 0F              ANDIM $0F    ORIGINAL UNIT/FCT
0080:  0228 C5 EF              CMP   TPFCT
0090:  022A 60                 RTS          EQUAL MEANS UNIT READY
0100:
0110:  022B A9 00      BELL    LDAIM $00
0120:  022D 8D 02 17           STA   PBD    ZERO FCT SETS TONE
0130:  0230 20 3C 02           JSR   DELAY  WAIT, RESET & EXIT
0140:
0150:  0233 A9 07      CTLOFF  LDAIM $07    BITS 0-2 TO O/P
0160:  0235 8D 03 17           STA   PBDD
0170:  0238 8D 02 17           STA   PBD    SET TO FCT#7 (OFF)
0180:  023B 60                 RTS
```

```
0190:
0200: 023C A9 FF      DELAY  LDAIM $FF
0210: 023E 8D 07 17          STA   $1707  SET TIMER TO 1/4 SEC
0220: 0241 2C 07 17          BIT   $1707
0230: 0244 10 FB             BPL   DELAY  +05
0240: 0246 60                RTS
0250:
0260: 0247 20 33 02  BRKCK   JSR   CTLOFF ENSURE OFF
0270: 024A 18                CLC
0280: 024B AD 02 17          LDA   PBD
0290: 024E 29 08             ANDIM $08    BIT 3 HIGH MEANS NO BRK
0300: 0250 D0 01             BNE   BKEXIT
0310: 0252 38                SEC
0320: 0253 60        BKEXIT  RTS          NO CARRY MEANS NO BRK
ID=04

0010:
0020:                *** CASSETTE WRITE ROUTINE ***
0030:
0040: 0254 D8        WRTAPE  CLD
0050: 0255 A9 01             LDAIM $01    TEST FOR UNIT#1 READY
0060: 0257 20 1B 02          JSR   TPTEST FOR WRITE?
0070: 025A F0 0C             BEQ   CWRITE ...YES
0080: 025C A9 03             LDAIM $03    ...NO, TEST UNIT#2
0090: 025E 20 1B 02          JSR   TPTEST
0100: 0261 F0 05             BEQ   CWRITE ...YES
0110: 0263 20 2B 02          JSR   BELL   ...NO, SOUND SIGNAL AND TRY
0120: 0266 D0 EC             BNE   WRTAPE AGAIN
0130:
0140: 0268 EA        CWRITE  NOP
0150:                .
0160:                .
0170:                . CASSETTE WRITE ROUTINE
0180:                . GOES HERE
0190:                .
0200:                .
0210: 0269 20 33 02          JSR   CTLOFF TURN OFF MOTORS
0220: 026C 4C 8C 1E          JMP   INIT   AND RETURN VIA KIM
ID=05

0010:                *** ALT.SW TEST & LIGHT ***
0020:
0030: 026F A9 06     TSTSWA  LDAIM $06    SET FOR ALT.SW #1
0040  0271 D0 02             BNE   TSTSWB +02
0050:
0060: 0273 A9 05     TSTSWB  LDAIM $05    SET FOR ALT.SW #2
0070: 0275 48                PHA          SAVE CODE
0080: 0276 20 33 02          JSR   CTLOFF INITL PORTS
0090: 0279 68                PLA          RETRIEVE CODE
0100: 027A 20 1B 02          JSR   TPTEST AND TEST SW
0110: 027D 18                CLC
0120: 027E D0 01             BNE   TSTX   IF NOT EQUAL
0130: 0280 38                SEC          MEANS SW IS NOT SET
0140: 0281 4C 33 02  TSTX    JMP   CTLOFF CARRY MEANS SW 'ON'
ID=
```

EXPAND YOUR 6052-BASED TIM MONITOR

Russell Rittimann
2606 Willow Crest
San Antonio, TX 78247

This modification to TIM will expand its command set such that ROM resident programs or routines can be executed from within TIM. Since I had several programs in PROM (BASIC, assembler, etc.) that were used regularly, I wanted an easy way to execute them without the usual sequence of: displaying the registers; setting the program counter; and finally typing "G". Now my TIM monitor will recognize a "B" from the keyboard and immediately put me into BASIC, and similarly recognize other commands for the other programs.

The TIM manual from MOS TECHNOLOGY included a complete listing of the monitor program. The sequence for recognizing a command in TIM was: output the prompting "."; read the command; look the command up in a table; and then execute the command by indirectly jumping to the address of the routine that corresponded to the command. This sequence of instructions is located from 708F(16) to 70B4(16) in the TIM monitor. All I needed to do is intercept the command and check it against my own table before letting TIM have its turn at it, which presented a problem since the TIM program is in ROM and can't be changed.

What I did was to disable TIM for a "window" of 16 locations from 7090(16) to 709F(16) and enable a DM8578 32 x 8 PROM at these same locations. Figure 1 shows the schematic for the PROM and address decoding. Note, that the 3-input NAND gate connected to CS2 of TIM, limits the monitor to between 6000(16) and 7FFF(16). This was not shown in the TIM manual.

I programmed the first half of the 8578 identical to the 16 locations in TIM starting at 7090(16) except for locations 4, 5, 6 (corresponding to TIM's 7094(16) - 7096(16).) In TIM, these 3 locations are a jump to subroutine to read a character from the keyboard. Instead, I put a jump to location CC00(16) where I had a 2708 EPROM decoded. The program in the 8578 is shown in Figure 2.

Figure 3 shows the program in the 2708. This instruction sequence receives the command from the keyboard and checks it against its command table. If not found, program control is returned to TIM at location 7098(16) to check its commands. If the command is user-defined, then the program jumps indirectly to the routine

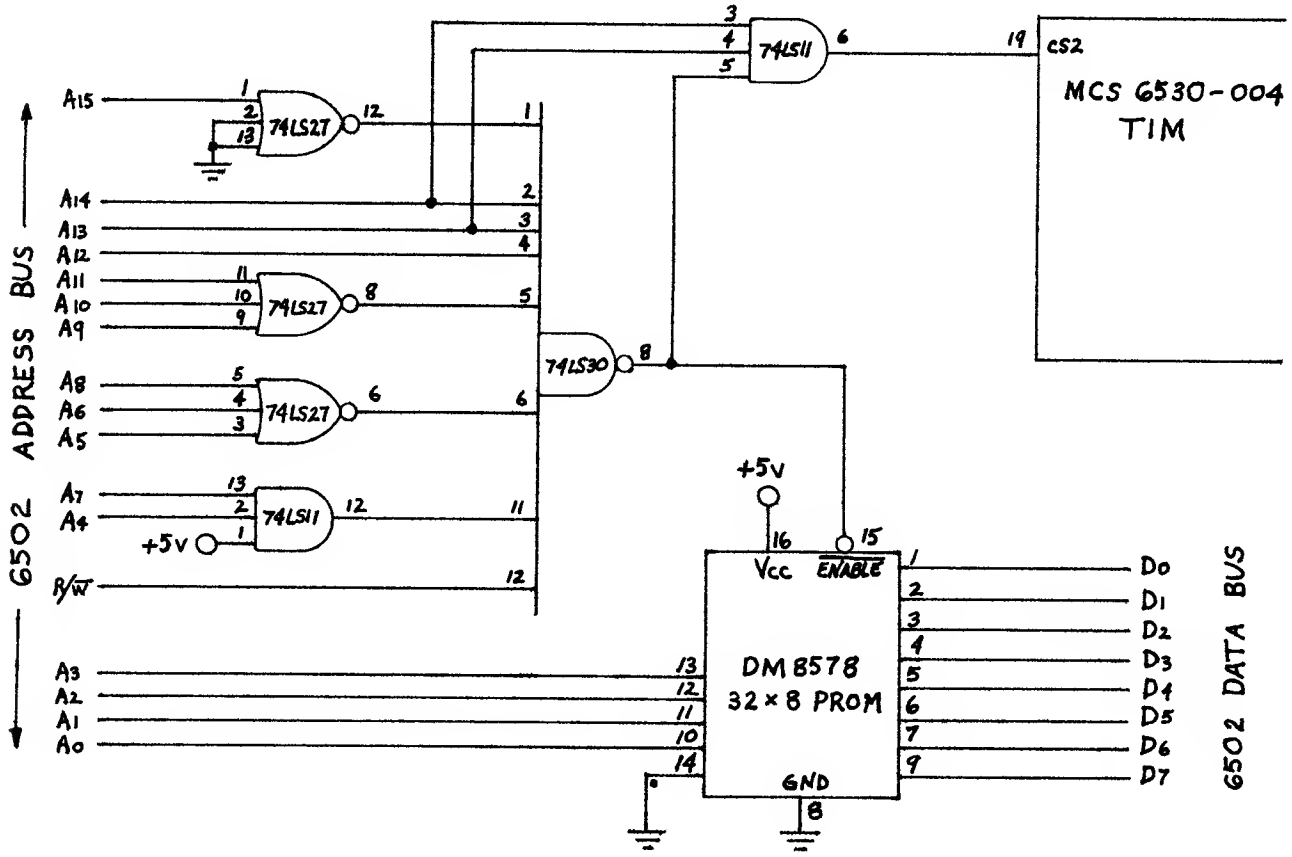


Figure 1
Schematic Diagram

```
LOC.    CONTENTS      INSTR.          COMMENTS

0000-   2E            .HS $2E         TIM PROMPTER "."
0001-   20 C6 72      JSR $72C6       OUTPUT PROMPTER USING TIM OUTPUT ROUTINE
0004-   4C 00 CC      JMP $CC00       JUMP INTO 2708 EPROM
0007-   A2 06         LDX #NCMDS-1    FOLLOWING INSTRUCTIONS AS IN TIM
0009-   DD 06 71      CMP CMDS,X
000C-   D0 19         BNE S2
000E-   A5 FD         LDA SAVX
```

Figure 2
Program in 8578 PROM

whose address is immediately following the command letter in the table. User-defined commands will have priority over TIM's commands. The format for each command in the table is as follows: command letter, low address of routine, high address of routine. Since the 2708 is erased to all 1's, I used FF(16) for the delimiter to signify the end of the table. Thus, the table can be added to at any time by programming 3 bytes.

Some final comments: I located the 2708 at CC00(16) but it can be located anywhere by changing the address in the 8578 and the address of the command table. At the end of each routine added, there must be a jump to 7086(16) to get back into TIM. The first byte of the 8578 is the TIM prompting character. If you want something other than the period, program any character you want into this location. Since the 8578 is an irreversible PROM, and I only used the first 16 locations, if you make a mistake in burning the PROM, the second half can be used by connecting the high address line, A4, to Vcc. Also, check the 2708 before the 8578 is wired since this modification won't work without all chips installed correctly.

This modification converts TIM into an adaptable operating system. Anytime I get more resident routines, I can add them to TIM by programming three locations into the command table in the 2708.

```
LOC.    CONTENTS      INSTR.          COMMENTS

CC00-   20 E9 72      JSR $72E9       GET COMMAND USING TIM INPUT ROUTINE
CC03-   A2 00         LDX #$00        X IS INDEX INTO TABLE
CC05-   BC 28 CC  LOOP LDY TABL,X     CHECK COMMAND LETTER IN TABLE FOR DEFAULT
CC08-   C0 FF         CPY #$FF            DELIMITER
CC0A-   D0 03         BNE CHEK        IF NOT DELIMITER, COMPARE COMMAND FROM KEYBOARD
CC0C-   4C 97 70      JMP $7097       OTHERWISE, JUMP BACK INTO TIM
CC0F-   DD 28 CC  CHEK CMP TABL,X     CHECK KEYBOARD COMMAND AGAINST TABLE
CC12-   D0 0F         BNE NEXT        IF NOT COMMAND, CHECK NEXT IN TABLE
CC14-   E8            INX             FOUND COMMAND
CC15-   BD 28 CC      LDA TABL,X      GET LOW ADDRESS OF ROUTINE
CC18-   85 EC         STA $EC
CC1A-   E8            INX
CC1B-   BD 28 CC      LDA TABL,X      GET HIGH ADDRESS OF ROUTINE
CC1E-   85 ED         STA $ED
CC20-   6C EC 00      JMP ($00EC)     JUMP INDIRECT TO ROUTINE
CC23-   E8        NEXT INX           INCREMENT POINTER TO NEXT COMMAND
CC24-   E8            INX
CC25-   E8            INX
CC26-   D0 DD         BNE LOOP        GO BACK AND CHECK REST OF COMMANDS
CC28-   2A        TABL .HS $2A        COMMAND LETTER "*"
CC29-   92             .HS $92        LOW ADDRESS OF ROUTINE #1
CC2A-   CC             .HS $CC        HIGH ADDRESS OF ROUTINE #1
CC2B-   42             .HS $42        COMMAND LETTER "B" FOR BASIC PROGRAM
CC2C-   A1             .HS $A1        LOW ADDRESS OF BASIC PROGRAM
CC2D-   CC             .HS $CC        HIGH ADDRESS OF BASIC PROGRAM
CC2E-   FF             .HS $FF        END OF TABLE DELIMITER
```

Figure 3
Program in 2708 EPROM

6502 GRAPHICS ROUTINES

Jim Green
807 Bridge Street
Bethlehem, PA 18018

The 6502 Graphics routines were written specifically for use with the Polymorphics Video Terminal Interface. (VTI), board installed on a KIMSI, S-100 interface to a KIM. It is expected that these routines will work with other low resolution graphics boards of a similar configuration with little or no software modification but no other boards have been attempted to date.

On the VTI, 16 lines of 64 ASCII characters each, or a grid of 48 by 128 individually controllable points, can be accomodated. For each memory byte, the high bit, 7, determines how the byte is to be treated. If this bit is set the byte is displayed as an ASCII character. If the bit is clear the lowest six bits are displayed as points of a 3 by 2 point subset of the 48 by 128 point grid. Each such bit that is set will be displayed as black. The remainder will be white.

The upper left-hand corner of the display screen is the display origin. This is also the base address of the video memory. The input coordinates to the routines are specified as hex values in the X and Y registers. The X register holds the column value and has a permitted range of 0 to $7F. The Y register holds the row value from 0 to $3F.

Routines are provided: WHTPNT, to set; BLKPNT, to clear; and TSTPNT, to test the current value of a specified display bit. An additional routine, BLANKR, is used to blank the entire screen.

The principal task of the graphics routines is to gain effective access to each specified bit in the screen grid without disturbing any of the remaining points. This task is divided into two parts, first to locate the byte that contains the target bit, then to isolate the target bit itself. These tasks are performed by subroutine POINT which is called by the other routines.

Within POINT, the ranges of X and Y are first tested. If either value is found to be out of range, control is returned to the calling program with the C flag set and with no changes to the video memory. If the ranges are ok, the C flag will be clear when the routines eventually return to the calling program.

After the range tests, the row coordinate value is divided by three (by a process of successive subtraction), and the column coordinate value is divided by two. The integer quotents and remainders are saved separately in each case. The row and column quotents now point to the row and column that uniquely contain the target byte. Hence, when the row quotent is multiplied by 64 and is then added to the column quotent an offset from the video memory base address is obtained. By adding the base address to this value the absolute memory address of the target byte is obtained.

To isolate the target bit within a byte, the column and row remainders are combined to form an index value (0 to 5). This index is used to select one of six masks which may be logically combined with the byte to uniquely treat the target bit.

On the system described, these routines require an average of about a third of a milli-second to complete a single bit update. This is more than ample for most purposes.

```
;       6502 GRAPHICS ROUTINES
;       VERSION 0.3B, 18 OCT 78
;
;       COPYRIGHT BY
;       J. S. GREEN, COMPUTER SYSTEMS
;       807 BRIDGE STREET
;       BETHLEHEM, PA  18018
;       (215) 867-0924
;
;       COMMERCIAL RIGHTS RESERVED
;
;       EFFECTIVE COORDINATES IN HEX ON ENTRY:
;             COLUMN VALUE IN X, (0 - $7F)
;             ROW VALUE IN Y,    (0 - $2F)
;
;       CONSTANTS
        .DEF  VIDBAS=$C000        ;VIDEO MEMORY BASE ADDR
        .DEF  UPLIM=$C4   ;UPPER LIMIT (HIGH BYTE)
;
```

```
                        ;       VARIABLES
                                .DEF   ROW=$E2
                                .DEF   COL=$E3
                                .DEF   ROWREM=$E4
                                .DEF   COLREM=$E5
                                .DEF   GRADR=$E6
                        ;
                                .LOC   $0200
                        ;
                        ;       DISPLAY CLEAR BIT
                        ;
0200 20 4C 02   WHTPNT: JSR    POINT       ;GET ADDRES + MASK INDEX
0203 B0 09              BCS    WHTPT1      ;BR IF PROBLEM
0205 A0 00              LDY#   0
0207 BD 90 02           LDAX   PLTMSK      ;GET MASK
020A 31 E6              AND@Y  GRADR       ;AND WITH VIDEO BYTE
020C 91 E6              STA@Y  GRADR       ;DISPLAY CLEAR BIT
020E 60         WHTPT1: RTS
                        ;
                        ;       DISPLAY SET BIT
                        ;
020F 20 4C 02   BLKPNT: JSR    POINT       ;GET ADDRES + MASK INDEX
0212 B0 0D              BCS    BLKPT1      ;BR IF PROBLEM
0214 A0 00              LDY#   0
0216 BD 90 02           LDAX   PLTMSK      ;GET MASK
0219 49 FF              EOR#   $FF         ;REVERSE IT
021B 11 E6              ORA@Y  GRADR       ;OR WITH VIDEO BYTE
021D 29 3F              AND#   $3F         ;CLEAR HIGH BITS
021F 91 E6              STA@Y  GRADR       ;DISPLAY SET BIT
0221 60         BLKPT1: RTS
                        ;
                        ;       TEST DISPLAYED BIT
                        ;       RESULTS WITH Z FLAG SET IF BIT IS SET
                        ;
0222 20 4C 02   TSTPNT: JSR    POINT       ;GET ADDRES + MASK INDEX
0225 B0 0B              BCS    TSTPT1      ;BR IF PROBLEM
0227 A0 00              LDY#   0
0229 BD 90 02           LDAX   PLTMSK      ;GET MASK
022C 49 FF              EOR#   $FF         ;REVERSE IT
022E 29 BF              AND#   $BF         ;CLEAR BIT 6
0230 31 E6              AND@Y  GRADR       ;Z SET IFF GRAPHIC-BIT SET
0232 60         TSTPT1: RTS
                        ;
                        ;       BLANK VIDEO FOR PLOT
                        ;
0233 A9 C0      BLANKR: LDA#   >VIDBAS
0235 85 E7              STA    GRADR+1
0237 A0 00              LDY#   0
0239 84 E6              STY    GRADR
023B A9 3F              LDA#   $3F          ; 0011 1111
023D 91 E6      BLANK1: STA@Y  GRADR
023F E6 E6              INC    GRADR
0241 D0 FA              BNE    BLANK1
0243 E6 E7              INC    GRADR+1      ;HIGH ORDER ADDRES BYTE
0245 A6 E7              LDX    GRADR+1
0247 E0 C4              CPX#   UPLIM        ;TEST END OF SCREEN
0249 90 F2              BCC    BLANK1       ;BR NOT DONE
024B 60                 RTS
                        ;
```

```
                        ;       GET BYTE ADDRES & BIT MASK
                        ;
024C E0 80      POINT:  CPX#    $80         ;128 IS TOO HIGH
024E B0 3F              BCS     POINT3      ;BR TOO HIGH
0250 C0 30              CPY#    $30         ;48 IS TOO HIGH FOR ROW
0252 B0 3B              BCS     POINT3      ;BR TOO HIGH
0254 8A                 TXA                 ;COLUMN
0255 48                 PHA                 ;SAVE IT
0256 98                 TYA                 ;ROW
0257 AA                 TAX                 ;DIVIDE ROW BY 3
0258 A0 FF              LDY#    $FF         ;INITIALIZE QUOTENT
025A C8         POINT1: INY                 ;ACCUMULATE QUOTENT
025B CA                 DEX                 ;SUBTRACT 3
025C CA                 DEX
025D CA                 DEX
025E 10 FA              BPL     POINT1      ;BR MORE
0260 E8                 INX                 ;RESTOR 3
0261 E8                 INX
0262 E8                 INX
0263 86 E4              STX     ROWREM      ;ROW REMAINDER
0265 84 E2              STY     ROW         ;INTEGER QUOTENT
0267 A2 00              LDX#    0
0269 86 E5              STX     COLREM      ;INITIALLY CLEAR
026B 68                 PLA                 ;RESTOR COLUMN
026C 4A                 LSRA                ;DIVIDE BY 2
026D 85 E3              STA     COL         ;INTEGER QUOTENT
026F 26 E5              ROL     COLREM      ;REMAINDER FROM CARRY
0271 A5 E2              LDA     ROW
0273 18                 CLC
0274 86 E7              STX     GRADR+1     ;CLEAR ADDRES HI
0276 A2 05              LDX#    5           ;PREP TO MPY BY 2**6 (=64)
0278 0A         POINT2: ASLA                ;MPY BY 2 EACH LOOP
0279 26 E7              ROL     GRADR+1     ;OVERFLO TO ADDRES HI
027B CA                 DEX
027C 10 FA              BPL     POINT2      ;BR TIL DONE 6 TIMES
027E 65 E3              ADC     COL         ;ADD THE PLACE IN THE ROW
0280 85 E6              STA     GRADR
0282 A9 C0              LDA#    >VIDBAS     ;VIDEO MEMORY BASE ADDR HI
0284 65 E7              ADC     GRADR+1
0286 85 E7              STA     GRADR+1     ;ADDRES POINTS TO BYTE
                                            ; IN VIDEO MEMORY
                        ;
                        ;       NOW CALC MASK INDEX FOR BIT WITHIN BYTE
0288 A5 E4              LDA     ROWREM      ;EITHER 0, 1 OR 2
028A 66 E5              ROR     COLREM      ;EITHER 0 OR 1 INTO CARRY
028C 2A                 ROLA                ;COMBINE WITH CARRY
028D AA                 TAX
028E 18                 CLC                 ;CLEAR CARRY SAYS ANS OK
028F 60         POINT3: RTS
                        ;
0290 1F         PLTMSK: .BYTE   $1F         ;UP-LEFT POINT WITHIN BYTE
0291 3B                 .BYTE   $3B         ;UP-RT
0292 2F                 .BYTE   $2F         ;MID-LF
0293 3D                 .BYTE   $3D         ;MID-RT
0294 37                 .BYTE   $37         ;LO-LF
0295 3E                 .BYTE   $3E         ;LO-RT
                        ;
                        .END                NO ERRORS DETECTED
                                            PASS (1-2)?
```

A CLOSE LOOK AT THE SUPERBOARD II

Bruce Hoyt, Pastor
Sharon Associated Reformed Presbyterian Church
Route 1
Brighton, TN 38011

Late in December 1978 my dreams came true. Those dreams I had had in the mid 60's when I first learned how to program computers. I had dreamed of having my own desk-sized computer. That dream has come true to a degree I would not have thought possible then. The computer I now have is not desk-sized but is contained on one printed circuit board. Furthermore it is more powerful than the big monsters I worked on in the mid 60's.

I don't want to bore you with a description of my continual amazement at a computer on a chip for such things are now old hat. Nor do I want to give just a general overview of the Superboard II manufactured by Ohio Scientific. For a general description you may check the March 1979 issue of **Popular Electronics**, p.76. I want to go somewhat deeper into evaluating and describing the Superboard II (Note: the Challenger IP also manufactured by Ohio Scientific is the same computer in a case with power supply).

HARDWARE

KEYBOARD:

The keyboard is mounted directly on the printed circuit board as can be seen in the advertisements. It is a polled keyboard which is polled by writing to a latch addressed at memory location DF00. This latch feeds the rows of the keyboard matrix. When a key is depressed the latch signal is fed through the key switch to a tri-state buffer and back onto the data buss. A read of address DF00 will pick up the signal from the column in which the key is depressed. This method of polling the keyboard makes the hardware very simple (and cheap) but it is effective. In my view a polled keyboard like the one on the Superboard II is better than a hardware implemented ASCII keyboard. Several nice features can be incorporated this way. For example every key has an automatic repeat feature. You have direct access to every key on the board for gaming purposes. Another keyboard can be put in parallel with the existing one. I plan to add a Hex keypad this way. OSI has provided a jack with several of the keyboard lines on it so that switch type joysticks may be connected for games.

For ordinary ASCII input from the keyboard the monitor includes a subroutine which returns the ASCII value of any key depressed. So for all practical purposes this arrangement works just like any other ASCII keyboard.

OSI has fed the signal from the keyboard through a resistor network and then out the game jack. This signal may be connected to a speaker to make sounds or music. The only reason I cannot give a further description of this feature is that OSI failed to include the resistors and I haven't yet gotten around to it.

VIDEO DISPLAY:

The video display is elegant and simple from a hardware point of view. The display on the screen is 32 by 32 but has no guard bands. My monitor displays about 27 by 30 screen size. The software supplied with the Superboard uses only 24 character lines since many who buy the Superboard may want to connect it to an ordinary TV through a video modulator. The video display is refreshed from a 1K memory located at D000-D3FF. Any byte written into this memory gets fed through a character generator and then sent to the screen. The character generator produces not only the full set of ASCII symbols but also more than 100 graphics symbols. It is complete enough to do just about anything you would want to on a 24 by 24 screen: Life, Tic-Tac-Toe, Pong, Racecar, Ship-tank-airplane warfare, etc.

You may wonder about the access to the refresh memory since both the CPU and the video disply circuitry must use it. The video display memory is accessed through a multiplexer which is normally connected to the refresh circuitry. This multiplexer allows the CPU to access the memory whenever the CPU addresses any memory from D000to D3FF. This causes a slight blink in the display on the TV monitor but the blink is almost unnoticable. Even constantly writing to the display memory causes only a slight decrease in brightness and some flicker of the picture. But whoever writes constantly to the display memory anyway? There is no affect at all on the monitor when the CPU is accessing memory other than the video memory.

CASSETTE I/O:

The Superboard comes with a KC standard cassette interface built in. This operates at 300 baud. That is somewhat slow for loading long programs but the slowness is compensated for by the accuracy. I have yet to find a read error. The hardware for the interface uses a Motorola 6850 ACIA to generate serial data. I think that a small change in the clock used for this ACIA could speed up operation but I have not checked this out yet. This 6850 is located at FOOOFOOI in the memory space.

The greatest difficulty with the cassette interface is that no provision has been made for motor control. It would have been simple to use the Request-to-Send output from the 6850 for this purpose. I plan to connect the Request-to-Send output to a small reed relay for this purpose.

COMPONENTS:

The board itself is high quality epoxy-glass. It is double sided, through the hole plated. The CPU is a 6502A and so has plenty of reserve. The RAM chips and other support are mostly low power variety. All have recent date codes. The character generator and the BASIC ROM's are masked programmed type but the monitor is an EPROM. I suppose you could reprogram the Monitor to suit some particular need you might have. The schematics are accurate and clear. They are very easy to follow since this computer is not really very complicated. The only complaint I would have is that various sections of the schematic are not labelled as to their funtion. But with a little study you can figure them out.

FUTURE EXPANSION:

An empty 40 pin DIP socket is provided for expansion. All the important control, address, and data lines are connected to this DIP socket. OSI makes a model 610 expansion board which connects to this DIP socket. The 610 expansion board comes with a timer, printer interface, and disk interface along with room for more memory. I personally plan to go from this DIP socket to a KIM type connector for interfacing but there are many possibilities for expansion including the S-100 bus or OSI's 48 pin bus.

SOFTWARE

MONITOR:

The monitor comes in an EPROM at the high end of memory and contains the interrupt vectors, the keyboard input routine, cassette I/O routines, and a memory access routine which allows you to view or change any memory location. With this capability it is very easy to load mchine language programs by hand and then execute them or save them and later load them from tape. One deficiency is the lack of a cassette save routine in the monitor.

The monitor has a load routine but no save routine. I have written a save routine which incorporates a Hex memory dump. (See figure 1) This routine saves data in a format acceptable to the monitor load routine. I have located it at 0222 since this space is unused by the BASIC interpreter. The begin address and the end address of the code to be saved must be entered at 00F7 and 00F9 respectively. When you execute the save routine, be sure to turn on your recorder! The code will be saved on tape as well as displayed on the monitor screen. If you want to use this program as a memory dump just run it without turning on your cassette. Several important monitor routines as well as some Basic routines are listed in Table 1.

BASIC:

The BASIC in ROM is an 8K Microsoft product. It is called a 6 digit BASIC since only 6 digits of precision are displayed. Internally, however, all numbers are carried in floating point form with 23 bits of precision (actually the precision is 24 bits since a high order 1 bit is assumed). That amounts to 7½ digits of precision internally. Though this BASIC is very good and very fast it is still a BASIC interpreter and allowance must be made for that fact. I have a puzzle that I have programmed in both BASIC and machine language. The machine language program takes about 1½ hours to run to completion. The BASIC program would take over a month! Superboard is what OSI calls its "immediate mode." That means that any statement can be entered without a line number and it will be executed immediately. Since "?" can be used in place of "PRINT" it is possible to interrogate the computer for any piece of information you might want. For example ? A yields thevalue of the variable A in the memory. ? 45-20 yields 25. ? PEEK (255) yields the contents of memory location 255 in decimal. GOTO 40 sends BASIC to statement number 40 and begins execution at that point. This last feature is very useful in debugging. One could say that the immediate mode allows you to use the Superboard as a super-calculator and provides a built-in debugger. The BASIC alone is worth the price of the computer.

ASSEMBLER:

There is one available from OSI on tape but I haven't tried it out. I want to write my own and put it in an EPROM.

DOCUMENTATION:

A few words must be said about documentation. Frankly, it is not up to OSI's high quality in the hardware and software areas. The graphics manual is by far the best, providing pretty clear descriptions and giving good examples. The users manual leaves something to be desired in clarity. It is too brief and rather vague at points. I have had real trouble trying to use machine language since there is virtually no description of the machine instructions. I also had some trouble figuring out what pins to connect my cassette to since the diagram is not clearly labelled. The BASIC manual is very brief–admittedly so. OSI expects you to have on hand a BASIC reference manual if you are not thoroughly familiar with the workings of BASIC. One serious problem is an error in the BASIC manual relating to the USR function. It tells you to poke the starting address of the USR routine into locations 023E-023F but this does not work. In the graphics manual there is an example of the use of the USR function. In that example the starting address of the USR routine is poked into 000B-000C. This works. I do wish that manufacturers would supply complete documentation with their software including source code. OSI provides almost nothing in the way of description for either the monitor or BASIC. I have disassembled the monitor and figured it out but have not yet started on BASIC. If anyone has inside information on the inner workings of Superboard BASIC please let us know. Think of all those good routines in BASIC that we could use to memory saving advantage: conversion routines, arithmetic routines, text editor, scanner, etc.

Though I have had to give a few negatives about the Superboard II I am well impressed with the quality of both hardware and software. If you are undecided as to what computer is the best buy for the money, I urge you to send your $279 check to OSI and ask for a Superboard. I don't think there is anything as good for the price on the market.

OSI CASSETTE SAVE/HEX MEMORY DUMP

BRUCE HOYT
MARCH 1979

TO USE, PLACE THE START ADDRESS OF CODE TO BE SAVED IN 00F7,00F8 AND THEN THE END ADDRESS IN 00F9,00FA. TURN ON THE TAPE RECORDER AND EXECUTE. NOTE: THIS PROGRAM WILL SAVE ITSELF ON TAPE.

```
0222                    ORG   $0222

0222 A9 0D      START   LDAIM $0D     CARRIAGE RETURN
0224 20 2D BF           JSR   $BF2D   CRT
0227 20 7A FF           JSR   $FF7A   10 NULLS TO CASSETTE
022A A9 2E              LDAIM $2E     "." ADDRESS MODE
022C 20 75 02           JSR   CC
022F A5 F8              LDA   $00F8   FROM LOCATION (HIGH)
0231 20 63 02           JSR   AOUT
0234 A5 F7              LDA   $00F7   FROM LOCATION (LOW)
0236 20 63 02           JSR   AOUT
0239 A9 2F              LDAIM $2F     "/" DATA MODE
023B 20 75 02           JSR   CC

023E A2 00      LOOP    LDXIM $00
0240 A1 F7              LDAIX $00F7   GET BYTE
0242 20 63 02           JSR   AOUT    OUTPUT
0245 A9 0D              LDAIM $0D     CARRIAGE RETURN
0247 20 B1 FC           JSR   $FCB1   CASSETTE OUTPUT
024A A9 20              LDAIM $20     SPACE
024C 20 2D BF           JSR   $BF2D   CRT
024F E6 F7              INC   $00F7   INCREMENT FROM ADDRESS
0251 D0 02              BNE   BUMP
0253 E6 F8              INC   $00F8
0255 38         BUMP    SEC           CHECK IF DONE
0256 A5 F9              LDA   $00F9   TO
0258 E5 F7              SBCZ  $00F7   FROM
025A A5 FA              LDA   $00FA   TO + 1
025C E5 F8              SBCZ  $00F8   FROM + 1
025E 10 DE              BPL   LOOP
0260 4C 43 FE           JMP   $FE43   YES, RETURN TO MONITOR

0263 85 FC      AOUT    STA   $00FC   USE MONITOR DISPLAY
0265 20 AC FE           JSR   $FEAC   TO UNPACK
0268 AD CC D0           LDA   $D0CC   HI
026B 20 75 02           JSR   CC
026E AD CD D0           LDA   $D0CD   LO
0271 20 75 02           JSR   CC
0274 60                 RTS

0275 20 B1 FC   CC      JSR   $FCB1   OUTPUT TO CASSETTE
0278 20 2D BF           JSR   $BF2D   AND CRT
027B 60                 RTS
```

Figure 1

Page 0 Usage

0000	JMP to warm start in BASIC
00FB	cassette/keyboard flag for monitor
00FC	data temporary hold for monitor
00FE-00FF	address temporary hold for monitor

Page 1

0100-0140	stack
0130	NMI vector - NMI interrupt causes a jump to this point
01C0	IRQ vector

Page 2

0200	cursor position
0203	load flag
0205	save flag
0206	CRT simulator baud rate - varies from 0 = fast to FF = slow
0212	Control-C flag
0218	input vector = FFBA
021A	output vector = FF69
021C	Control C check vector = FF9B
021e	load vector = FF8B
0220	save vector = FF96
0222-02FA	unused

Page 3 and up to end of RAM is BASIC workspace

A000-BFFF	BASIC in ROM
D000-D3FF	Video refresh memory
DF00	Polled keyboard
F000-F001	Cassette port 6850
F800-FFFF	Monitor EPROM
FC00	Floppy bootstrap
FD00	Keyboard input routine
FE00	Monitor
FF00	BASIC I/O support

Useful Subroutine entry points

A274	warm start for BASIC
BD11	cold start for BASIC
BF2D	CRT simulator - prints char in A register
FD00	input char from keyboard, result in A
FCB1	output 1 byte from A to cassette
FE00	entry to monitor, clears screen, resets ACIA
FE0C	entry to monitor, bypasses stack initialization
FE43	entry to address mode of monitor
FE80	input ASCII char from cassette, result in A, 7 bit cleared
FE93	convert ASCII hex to binary, result in A, =80 if bad
FF69	BASIC output to cassette routine, outputs one char to cassette, displays on screen, outputs 10 nulls if carriage return character
FF00	Reset entry point
FF8B	Load flag routine
FF96	Save flag routine
FF9B	Control-C routine
FFBA	BASIC input routine

Table 1.

TWO SHORT TIM PROGRAMS

Gary L. Tater
7925 Nottingham Way
Ellicott City, MD 21043

A Fast Talking TIM

If you have used both KIM and TIM with a terminal, you know that TIM has many nice features. For instance you can enter eight bytes at a time with TIM, and TIM has many more subroutines you can call in your programs than KIM does. However, KIM can adapt to terminal frequencies up to 2400 baud whereas TIM was designed to work from 100 to 300 baud. This article describes a program which allows you to communicate with TIM at 1200 baud or higher.

After a reset TIM automatically measures the speed of your terminal and deposits the bit times representative of the baud rate in two zero page locations, OOEA and OOEB. To increase the baud rate above 300 baud, the procedure is to place the correct values into EA and EB and change your terminal to that speed.

```
0100 20 A4 73  NEWVAL JSR   $73A4  READ TWO BYTES VIA TIM MONITOR
0103 A5 EE            LDA   $00EE  PUT EE INTO EB
0105 85 EB            STA   $00EB
0107 A5 EF            LDA   $00EF  PUT EF INTO EA
0109 85 EA            STA   $00EA
010B 00               BRK
010C 4C 00 01         JMP   NEWVAL TYPE G FOR NEW VALUES
```

Figure 1
Program to Change OOEA and OOEB.
Type Major Value OOEA First

By using the short program of Figure 1, I was able to find the correct values for 600 and 1200 baud operation (See Table 1) for my CT-64 and CGRS CPU board which has a 6502 operating with a one megahertz crystal. For each baud rate there is a range of values that is acceptable for EB. I have attempted to find the center of the range for my system. You will probably need to experiment to find the best numbers for your computer.

Baud Rate	00EA	00EB
1200	01	50
600	03	13
300	06	3C

Table 1
Zero page memory values for three baud rates.

Using this basic information I wrote the program of Figure 2. The programs begins at 157E and asks:

SPEED 300 600 1200?

At this point you should type 3, 6, or 1 and change your terminal to the correct rate. The program determines what you have entered and stores the correct values in EA and EB. By inspection of the program, you should be able to expand it to 2400 baud if you have a faster terminal. For a one megahertz system typical values are 00 in EA and 75 in EB for 2400 baud.

A TIM Operating System Menu

If you have written a collection of utility programs, assemblers, disassemblers and application programs, you will need a directory program with which you can easily call your desired program. The short program in Figure 3 uses the alphabet to call 26 programs. When the programs finish, they should return to the beginning of the directory program at location 0100.

You may choose to keep the program in ROM as I do. Only locations 0116 and 011B need be changed to do this provided you start the program at the beginning of a page.

The program prints a prompting "-" so that you'll know its in command and not TIM. If you type a nonalphabetic character, it will restart. After you type a letter, say a C for compare or M for move, the program finds the appropriate starting address stored between 0122 and 0155. After the starting address is stored in 00F6 and 00E7, the program calls the "GO" subroutine in TIM which causes your program to be executed.

```
          THIS PROGRAM IS RELOCATABLE AS LONG AS THE POINTER TO
          THE TEXT MESSAGE IS CHANGED IN LINE "PRINT"

157E D8          START  CLD           CLEAR DECIMAL MODE
157F A0 00             LDYIM $00      INITIALIZE INDEX
1581 B9 B3 15    PRINT  LDAY  TEXT    GET ASCII CHARACTERS
1584 F0 06             BEQ   PDONE    DONE IF NULL CHARACTER
1586 20 C6 72          JSR   $72C6    PRINT VIA TIM OUTPUT ROUTINE
1589 C8                INY            BUMP POINTER
158A D0 F5             BNE   PRINT    UNCONDITIONAL BRANCH TO PRINT NEXT
158C 20 E9 72    PDONE  JSR   $72E9    READ CHOICE VIA MONITOR
158F C9 31             CMPIM '1       ASCII 1 ?
1591 F0 1A             BEQ   HIGH     1200 BAUD
1593 C9 36             CMPIM '6
1595 F0 10             BEQ   MEDIUM
1597 C9 33             CMPIM '3
1599 D0 E3             BNE   START    NOT VALID CHARACTER

159B A2 3C       LOW    LDXIM $3C      GET VALUES FOR 300 BAUD
159D A9 06             LDAIM $06

159F 85 EA       FIXIT  STA   $00EA    SAVE FOR TIM TIMING ROUTINES
15A1 86 EB             STX   $00EB    SAVE SECOND VALUE
15A3 00                BRK            RETURN TO MONITOR
15A4 18                CLC            CLEAR CARRY
15A5 B0 D7             BCS   START    UNCONDITIONAL BRANCH

15A7 A2 13       MEDIUM LDXIM $13      GET VALUES FOR 600 BAUD
15A9 A9 03             LDAIM $03
15AB D0 F2             BNE   FIXIT    UNCONDITIONAL BRANCH TO FIXIT

15AD A2 50       HIGH   LDXIM $50      GET VALUES FOR 1200 BAUD
15AF A9 01             LDAIM $01
15B1 D0 EC             BNE   FIXIT    UNCONDITIONAL BRANCH TO FIXIT

15B3 53          TEXT   =     'S        "SPEED  300 600 1200 ?"
15B4 50                 =     'P
15B5 45                 =     'E
15B6 45                 =     'E
15B7 44                 =     'D
15B8 20                 =     '
15B9 20                 =     '
15BA 33                 =     '3
15BB 30                 =     '0
15BC 30                 =     '0
15BD 20                 =     '
15BE 36                 =     '6
15BF 30                 =     '0
15C0 30                 =     '0
15C1 20                 =     '
15C2 31                 =     '1
15C3 32                 =     '2
15C4 30                 =     '0
15C5 30                 =     '0
15C6 20                 =     '
15C7 3F                 =     '?
15C8 20                 =     '
15C9 00                 =     $00
```

Figure 2
6502 Program to Change Speed

```
0100 20 8A 72  START  JSR   $728A  CRLF VIA TIM MONITOR
0103 A9 2D            LDAIM '-     PRINT "-"
0105 20 C6 72         JSR   $72C6  VIA TIM MONITOR
0108 20 EE 72         JSR   $72EE  READ A CHARACTER VIA TIM
010B C9 5B            CMPIM $5B    TEST FOR GREATER THAN Z
010D 10 F1            BPL   START  BRANCH IF TOO LARGE
010F 38               SEC          SET TO CONVERT ASCII TO INDEX
0110 E9 41            SBCIM 'A     BY SUBTRACTING VALUE OF ASCII A
0112 30 EC            BMI   START  IF MINUS, THEN CHARACTER LESS THAN A
0114 0A               ASLA         MULTIPLY BY TWO FOR INDEX
0115 AA               TAX          PUT CONVERTED VALUE INTO INDEX
0116 BD 24 01         LDAX  LOWADR GET START ADDRESS LOW
0119 85 F6            STA   $00F6  SAVE FOR TIM
011B BD 25 01         LDAX  HGHADR GET START ADDRESS HIGH
011E 85 F7            STA   $00F7  SAVE START ADDRESS HIGH
0120 20 5C 71         JSR   $715C  GO TO SUBROUTINE VIA TIM
0123 00               BRK

0124 00        LOWADR =     $00    LOW ADDRESS FOR A, FILLED IN BY USER
0125 00        HGHADR =     $00    HIGH ADDRESS FOR A, FILLED IN BY USER
0126 00               =     $00    LOW ADDRESS FOR B
0127 00               =     $00    HIGH ADDRESS FOR B
               AND SO FORTH THROUGH
               LOW AND HIGH PAIR FOR Z
```

A 100 uS 16 CHANNEL ANALOG TO DIGITAL CONVERTER FOR 65XX MICROCOMPUTER SYSTEMS

J. C. Williams
55 Holcomb St.
Simsbury, CT 06070

Analog to digital (A/D) conversion can be useful in many microcomputer systems. The design presented here takes advantage of a large scale integrated circuit, the ADC0817, to simplify a 16 channel, 8 bit A/D system which can be attached to the bus of 65XX microcomputers. The applications that I have found for this system have included "straight" data acquisition, game joystick position reading, graphic input generation and voice recognition. Of course, the software for each of these applications is different, but they all require multichannel, reasonably fast A/D. The 100 us conversion time of this system depends only on the 1 MHz clock frequency of the microcomputer. The microprocessor is not involved in the A/D conversions. Once the conversion is started, the processor can work on other tasks until the digital result is available.

The Hardware

This device appears to the programmer as a block of memory starting at a base address, BASE, and extending through 16 locations to BASE † 15. (The actual circuit described occupies 256 locations because of incomplete decoding.) An analog to digital conversion of a selcted channel, say channel X, is started by writing to BASE † X. The 8 bit conversion result may then be read from any location in the block (eg. BASE) any time after the 100 US conversion time has elapsed. If desired, the end of conversion signal from the ADC0817 may cause an interrupt to get the attention of the processor. If multiple A/D conversions at the maximum speed are required the 65XX can be kept busy with "housekeeping" during the conversion delay time. The example programs illustrate two ways the converter may be driven. The system uses just five integrated circuits and can be built for less than $40. The design, shown in Figure 1, occupies a six square inch area on a Vector plugboard and draws only 60 mA of current from the †8 Volt DC unregulated power supply. Operation of the circuit is simple because the ADC0817 performs all analog switching and A/D functions. The base address of the converter is fixed by six switches attached to the DM8131 six bit comparitor. When the processor accesses memory locations having address bits A15-A10 matching the switch settings, the DM8131 output goes low. This output is NOR'ed with A9 and A8 to further reduce the memory space occupied by the circuit to one 65XX page. The possible base addresses which can be obtained with this decoder can fall on any 1K boundary and A9 and A8 must be "0's". For example, base addresses (in hex) can be set to A000 or A400 but not A100, A200, or A300. In the design drawn, A9 and A8 must be low for the A/D to be selected, but this could be changed if A9 and/or A8 were inverted using unused sections of the 74LS05. When the A/D is selected,the output of the NOR gate (pin 12 of the 74LS27) goes to a "1"; this can be used as a "board selected" signal if needed (eg. by KIM-1 users for DECODE ENABLE). The microprocessor R/W and 02 lines, along with an inverted board select signal and combined in two NOR gates which 1) latch channel select bits A3-A0 and start A/D conversion during 02 of write cycles and 2) enable the tri-state data bus drivers during 02 of

read cycles. The end of conversion (EOC) signal, produced by the ADC0817 when the most recent conversion has been completed, can be connected to a processor interrupt line through one of the 74LS05 open collector inverters. These interrupts must be cleared by starting another A/D conversion.

Wire-wrap construction is suitable for the circuit and component layout is not critical. It is good practice, however, to orient the analog input area away from digital circuits. The REF + and REF- reference voltages must not be noisy if the full accuracy, 20 mV per bit, is to be achieved. The +5 Volt regulator should not be shared with other circuitry. The layout used in one of the prototypes is sketched in Figure 2. Figure 2 also shows several input connections which may be useful. The circuit has two limitations: 1) input voltages must be between 0 and +5 Volts and 2) signals being converted should not change appreciably during the 100 us conversion period. Both of these limitations may be eliminated by appropriate analog conditioning circuitry, but the simplicity of the design is lost. Builders who want to add features to the circuit should consult the ADC0817 specification and application information.

The Software

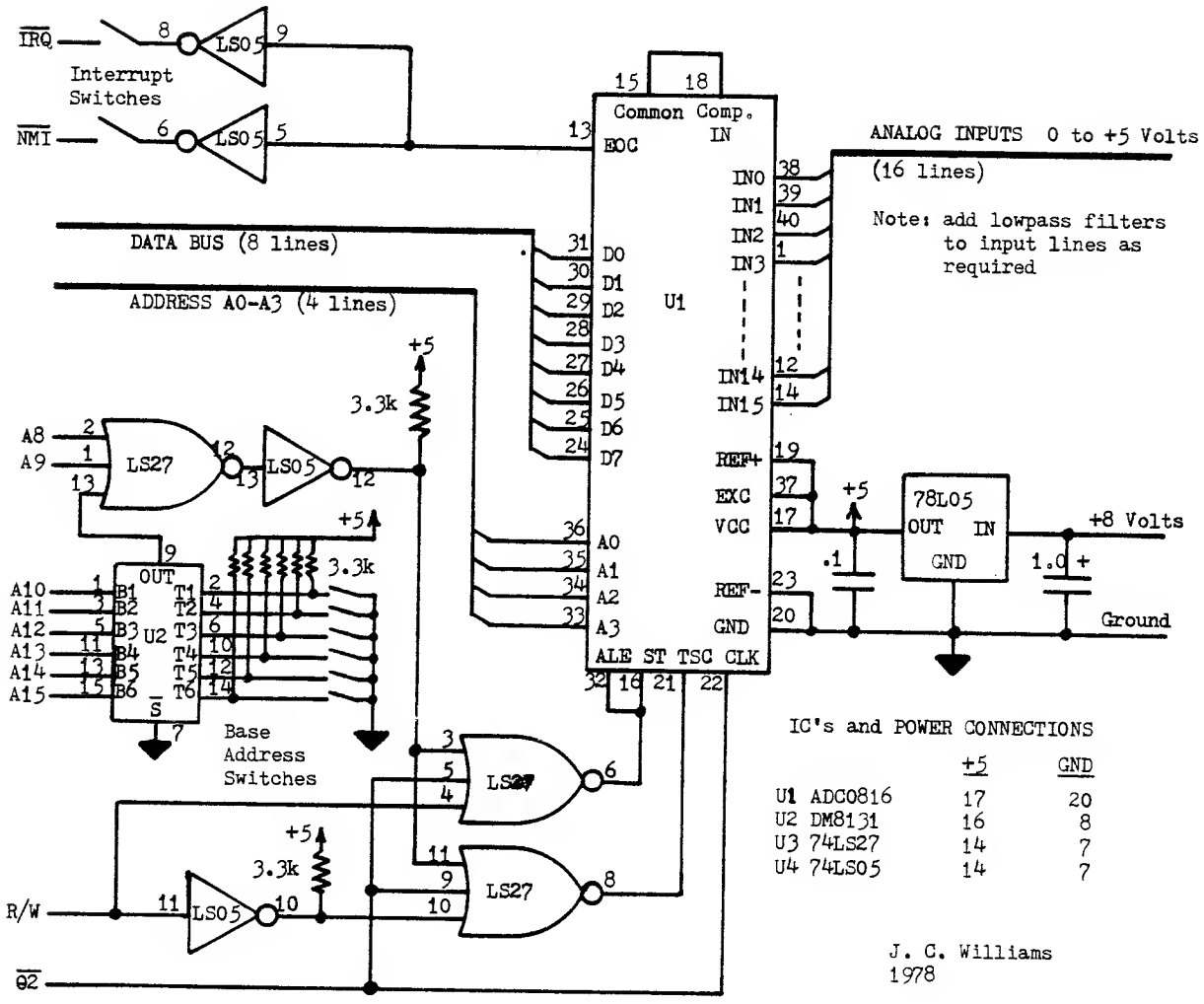
Two example subroutines which use the A/D converter illustrate how it is handled by software. The program which calls the A/D subroutine must initialize both the channel selection and storage defining parameters before the JSR instruction is executed. In the examples, an index register contains the channel selection information because of the ease of using an indexed addressing mode to start a conversion. Data storage is either on page 0 or pointed to by page 0 variables. The A/D subroutines must either contain delays or take enough time between writing to and reading from the ADC0817 to allow it to finish the conversion. Components for this very useful piece of hardware can be obtained from a number of sources readily available to low-volume users. Both National Semiconductor and Texas Instruments produce the ADC0817 and its more accurate counterpart, the ADC0816. The ADC0817 and its data sheet have been recently listed by TRI-TEK, Inc., 7808 N. 27th Ave., Phoenix, AZ 85021. Many other suppliers, such as Jameco Electronics, 1021 Howard Avenue., San Carlos, CA 94979, and Advanced Computer Products, 1310 "B" E. Edinger, Santa Ana, CA 92713, can supply the other components.

```
                        MCAD - MULTI-CHANNEL A/D CONVERSION
                        J. C. WILLIAMS
                        JANUARY 1979

0200                            ORG   $0200

0200                    BASE    *     $B000   BASE ADDRESS OF ADC0816
0200                    STORE   *     $9000   START OF 16 BYTE STORAGE AREA

0200 9D 00 B0           MCAD    STAX  BASE    START CONVERSION ON CHANNEL X
0203 A0 0E                      LDYIM $0E     DELAY FOR CONVERSION,
0205 88                 DY      DEY           MINIMUM VALUE = $0E
0206 D0 FD                      BNE   DY
0208 AD 00 B0                   LDA   BASE    GET CONVERTED DATA
020B 9D 00 90                   STAX  STORE   STORE DATA
020E CA                         DEX
020F 10 EF                      BPL   MCAD    DO NEXT CHANNEL
0211 60                         RTS           FINISHED

                        EXAMPLE CALLING ROUTINE FOR MCAD

0212 A2 0F              MCMAIN  LDXIM $0F     SELECT CONVERSION OF ALL
0214 20 00 02                   JSR   MCAD    16 CHANNELS AND GO TO SUBROUTINE
0217 00                         BRK           EXIT ** BE SURE TO INIT IRQ VECTOR **
```

```
                CXAD SUBROUTINE
                J. C. WILLIAMS
                JANUARY 1979

0300                    ORG   $0300

0300            BASE    *     $B000  BASE ADDRESS OF ADC0816
0300            SP      *     $0000  STORAGE POINTER
0300            SPSTR   *     $0002  LOC OF STORAGE BLOCK START ADDRESS
0300            SPSTP   *     $0004  LOC OF STORAGE BLOCK END ADDRESS

0300 9D 00 B0   CXAD    STAX  BASE   START FIRST CONVERSION
0303 A5 02              LDAZ  SPSTR  INIT STORAGE POINTER
0305 85 00              STAZ  SP
0307 A5 03              LDAZ  SPSTR  +01
0309 85 01              STAZ  SP     +01
030B D8                 CLD          USE BINARY MODE
030C A0 05              LDYIM $05    INSERT DELAY TO ALLOW
030E 88         DY      DEY          INITIAL CONV. TO COMPLETE
030F D0 FD              BNE   DY
0311 F0 16              BEQ   DELAY
0313 A5 00      TSTEND  LDAZ  SP     TEST FOR END OF
0315 C5 04              CMPZ  SPSTP  STORAGE BLOCK
0317 A5 01              LDAZ  SP     +01
0319 E5 05              SBCZ  SPSTP  +01
031B B0 1D              BCS   RT
031D A9 01              LDAIM $01    ADD ONE TO STORAGE POINTER
031F 65 00              ADCZ  SP
0321 85 00              STAZ  SP
0323 A9 00              LDAIM $00
0325 65 01              ADCZ  SP     +01
0327 85 01              STAZ  SP     +01
0329 A0 05      DELAY   LDYIM $05    DELAY TO FIX TIME BETWEEN CONV'S.
032B 88         DYA     DEY
032C D0 FD              BNE   DYA
032E AD 00 B0           LDA   BASE   READ CONVERTED RESULT
0331 9D 00 B0           STAX  BASE   START NEXT CONVERSION IMMEDIATELY
0334 A0 00              LDYIM $00    SET STORAGE OFFSET
0336 91 00              STAIY SP     STORE RESULTS
0338 F0 D9              BEQ   TSTEND ALWAYS TAKEN
033A 60         RT      RTS

                EXAMPLE CALLING ROUTINE FOR CXAD

033B A2 00      CXMAIN  LDXIM $00    SELECT CHANNEL 0
033D A9 00              LDAIM $00    SET STARTING ADDRESS OF
033F 85 02              STAZ  SPSTR  STORAGE BLOCK TO $9000
0341 A9 90              LDAIM $90
0343 85 03              STAZ  SPSTR  +01
0345 A9 FF              LDAIM $FF    SET ENDING ADDRESS OF
0347 85 04              STAZ  SPSTP  STORAGE BLOCK TO $9FFF
0349 A9 9F              LDAIM $9F
034B 85 05              STAZ  SPSTP  +01
034D 20 00 03           JSR   CXAD
0350 00                 BRK          EXIT  ** BE SURE TO INIT IRQ VECTOR **
```

FIGURE 1

16 CHANNEL ANALOG TO DIGITAL CONVERTER SYSTEM
FOR 65XX MICROPROCESSOR SYSTEMS



IC's and POWER CONNECTIONS

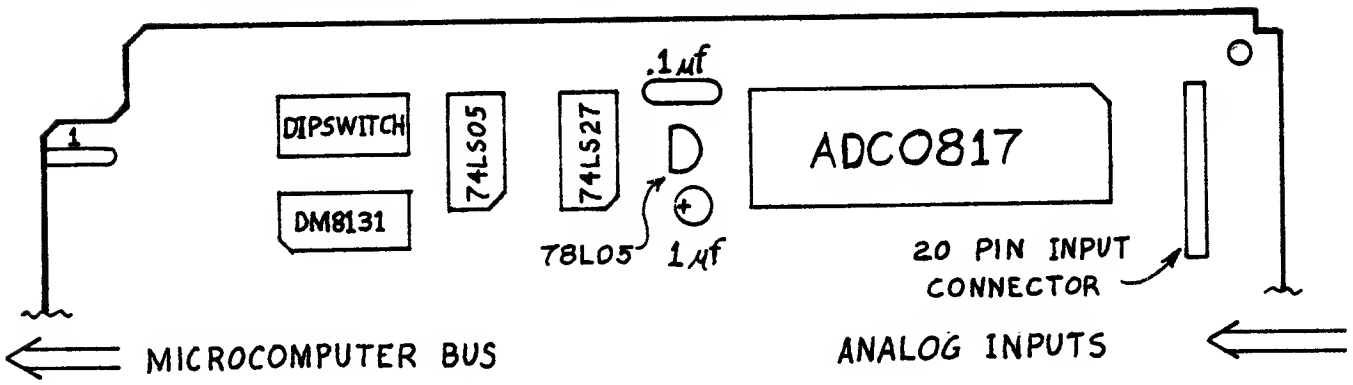
	+5	GND
U1 ADC0816	17	20
U2 DM8131	16	8
U3 74LS27	14	7
U4 74LS05	14	7

J. C. Williams
1978

FIGURE 2

16 CHANNEL A/D CONVERTER FOR 65XX SYSTEMS

COMPONENT SIDE OF 6.5" X 4.5" PROTOTYPING CARD - VECTOR 3662

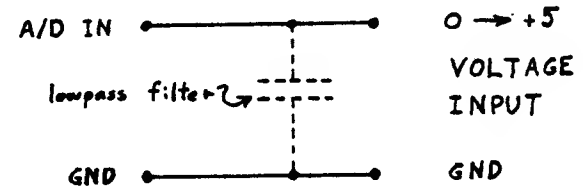


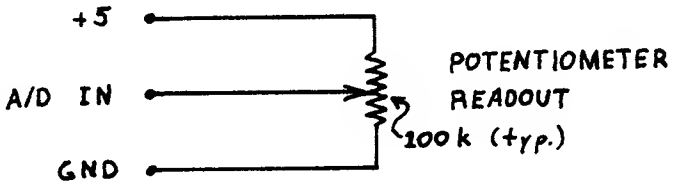
INPUT CONNECTOR DETAIL

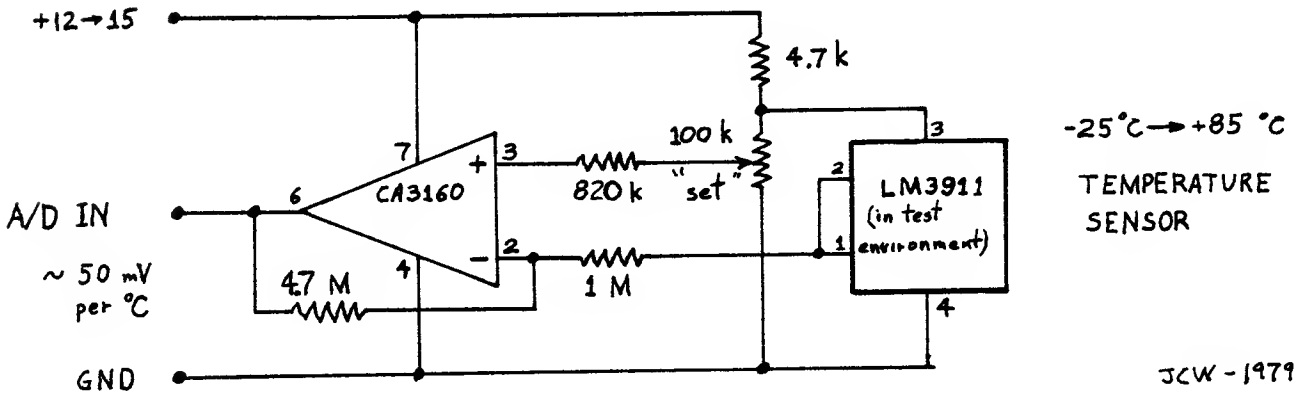
TOP VIEW

GND	• •	+5
IN14	• •	IN15
IN12	• •	IN13
IN10	• •	IN11
IN8	• •	IN9
IN6	• •	IN7
IN4	• •	IN5
IN2	• •	IN3
IN0	• •	IN1
GND	• •	+5

APPLICATIONS







USING TINY BASIC TO DEBUG MACHINE LANGUAGE PROGRAMS

Jim Zuber
20224 Cohasset No. 16
Canoga Park, CA 91306

I just got Tiny BASIC up and running on my KIM-1 and have found it to be a valuable enhancement to writing machine (or assembly) language programs. The Tiny BASIC USR function allows us to access machine language subroutines from within a BASIC program. You can pass parameters to and from the BASIC program and the machine language subroutine. If you can make an entire machine language program appear as a subroutine to Tiny BASIC (add a RTS call in the appropriate place) then Tiny BASIC can access your entire program with the USR function. A natural application of this capability is a debugging program written in Tiny BASIC that can completely test a machine language subroutine without ever leaving the Tiny BASIC program. The only limitation to this is that your machine language program cannot reside in the memory area used by Tiny BASIC and you must not use the same zero page locations as Tiny BASIC. My program (see listing #1) will print out the data in 4 memory locations when a predefined set of conditions exist in the machine language subroutine. There are 7 user selectable functions in the command mode:

(0) DEFINE SUBROUTINE ADDRESS - This is the starting address of the machine language subroutine you want to test.

(1) DEFINE PRINT ADDRESS - This allows you to define the conditions that must exist before data is printed out in the run mode. There are 3 options:
(A) Print every loop through the subroutine.
(B) Print at a predefined loop interval (use a decimal number).
(C) Conditional print - Program will only print out when data in a predefined address matches the value specified.
The print mode is initialized at "Print every loop" at the start of the program.

(3) PRESET DATA - This allows the user to place data in any memory location.

(4) PRINT LIMIT - This number limits the number of times the program prints out the 4 addresses when in the run mode. This is initialized at 10 in the beginning of the program. Use a decimal value for the print limit.

(5) RUN PROGRAM - This starts the Tiny BASIC program looping through and printing out data from the machine language subroutine.

(6) EXIT PROGRAM - Returns you to Tiny BASIC monitor.

(7) See COMMAND OPTIONS.

All command options except 6 return to the command mode after execution. If your version of Tiny BASIC does not start at hex 2000 you must change line 50 to the decimal equivalent of your Tiny BASIC starting address. All address and data questions should be answered in hex with a comma between each digit. Example: Address 0,2,0,0 or Data A,2. The program will print a marker every 50 loops through the machine language subroutine. This can be changed to suit your preference by modifying line 295. The following example should clarify the functions and use of the Debug Program. Listing #2 is a subroutine from a Biorhythm Program that I wrote. The subroutine increments 3 memory locations that correspond to the physical, emotional, and intellectual biorhythm cycle days. Each memory location should be reset to day one at the appropriate point.

Location 0001 = 1 to 23 days (physical)
Location 0002 = 1 to 28 days (emotional)
Location 0003 = 1 to 33 days (intellectual)

Two days in each cycle are considered critical. They are:

Physical: Day 1 and day 12
Emotional: Day 1 and day 15
Intellectual: Day 1 and day 17

Location 0004 is incremented for each cycle that is critical on a particular pass through the subroutine. An 02 in this location would indicate a double critical day on that particular pass through the subroutine. As a subroutine of this type would take several hours to test using conventional methods due to the large number of variables, the following sample runs from the Tiny BASIC machine Debug Program will show how complete testing of a subroutine can be done in a few minutes. (See samples #1 through #4).

Sample #1 The starting address is set to 0200 and the print addresses are defined as 0001 through 0004. The printout shows that locations 0001 - 0003 are incrementing as they should.

Sample #2 Memory locations 0001 - 0003 are preset to day 20 and the print limit is set to 15. The printout shows that the cycles are resetting to day 1 at the appropriate time. (days 23, 28, and 33).

Sample #3 The print limit is set to 4 and the print mode is set to every 23rd loop in order to check the consistency of the subroutine. The printout shows that location 0001 is staying the same as it should. (location 0001 is the 23 day physical cycle). Note the marker at 50 loops.

Sample #4 The print mode is set to the conditional mode in order to print only when location 0004 is equal to 02 (double citical day). The printout shows the subroutine is working properly.

I would like to thank Tom Pittman (author of Tiny BASIC) whose programming tricks in the Tiny BASIC User Manual made this program possible. I hope the Machine Debug Program can take the sweat out of testing your subroutines!

---------LISTING #1---------

```
10 REM TINY BASIC MACHINE DEBUG PROGRAM
11 REM BY JIM ZUBER---SEPT 29,1978
15 A=-10
20 B=-11
25 C=-12
30 D=-13
35 E=-14
40 F=-15
47 REM TINY START ADDRESS(DEC)
50 S=8192
54 REM PRESET PRINT LIMIT
55 G=10
59 REM PRESET PRINT MODE
60 H=1
65 REM ANSWER ALL ADDRESS AND DATA
70 REM QUESTIONS WITH A HEX NUMBER
75 REM THAT HAS EACH DIGIT SEPERATED
80 REM BY A COMMA.
85 REM AT A MINIMUM SET SUB ADDRESS
90 REM AND 4 DATA ADDRESSES.
100 PR "COMMAND MODE----SELECT ONE"
102 PR "   0.DEFINE SUBROUTINE ADDRESS"
103 PR "   1.DEFINE PRINT ADDRESSES"
106 PR "   2.DEFINE PRINT MODE"
109 PR "   3.PRESET DATA"
112 PR "   4.PRINT LIMIT"
115 PR "   5.RUN PROGRAM"
118 PR "   6.EXIT PROGRAM"
119 PR "   7.SEE COMMAND OPTIONS"
121 INPUT L
123 IF L=0 GOTO 275
124 IF L=1 GOTO 148
127 IF L=2 GOTO 187
130 IF L=3 GOTO 239
133 IF L=4 GOTO 266
136 IF L=5 GOTO 281
139 IF L=6 GOTO 1000
141 IF L=7 GOTO 102
142 PR "---COMMAND MODE---"
145 GOTO 121
147 REM INPUT 4 ADDRESSES
148 PR "POSITION #1--";
151 GOSUB 800
154 T=N
157 PR "POSITION #2--";
160 GOSUB 800
163 U=N
166 PR "POSITION #3--";
169 GOSUB 800
172 V=N
175 PR "POSITION #4--";
178 GOSUB 800
181 W=N
184 GOTO 142
186 REM DEFINE PRINT MODE
187 PR "PRINT MODE---SELECT ONE"
190 PR "   1.ALL LOOPS"
193 PR "   2.DEFINE NUMBER OF LOOPS"
196 PR "   3.CONDITIONAL PRINT"
199 INPUT H
202 IF H=1 GOTO 142
205 IF H=2 GOTO 215
208 IF H=3 GOTO 224
211 GOTO 187
215 PR "INPUT LOOP INCREMENT"
218 INPUT K
221 GOTO 142
223 REM CONDITIONAL PRINT MODE
224 GOSUB 800
227 I=N
230 GOSUB 850
233 J=N
236 GOTO 142
238 REM PRESET DATA
239 GOSUB 800
242 P=N
245 GOSUB 850
248 Q=N
251 Y=USR(S+24,P,Q)
254 PR "ANY MORE TO PRESET?(1=N 2=Y)"
257 INPUT Y
260 IF Y=2 GOTO 239
263 GOTO 142
265 REM SET PRINT LIMIT
266 PR "INPUT PRINT LIMIT"
269 INPUT G
272 GOTO 142
274 REM DEFINE SUB START ADDRESS
275 GOSUB 800
276 O=N
278 GOTO 142
280 REM RUN PROGRAM
281 PR
284 PR "LOOP","-1-","-2-","-3-","-4-"
287 PR
289 X=0
290 P=0
293 P=P+1
294 Y=USR(O)
295 IF P=P/50*50 THEN PR P
296 IF H=1 GOTO 314
299 IF H=2 GOTO 323
301 REM CONDITIONAL PRINT
302 Y=USR(S+20,I)
305 IF J=Y GOSUB 500
308 IF X=G GOTO 142
```

```
311 GOTO 293
313 REM PRINT ALL LOOPS
314 GOSUB 500
317 IF X=G GOTO 142
320 GOTO 293
322 REM DEFINED NUMBER OF LOOPS
323 IF P=P/K*K GOSUB 500
325 IF X=G GOTO 142
327 GOTO 293
349 REM SUB TO PRINT 2 HEX DIGITS
350 M=Z/16
355 Z=Z-M*16
360 GOSUB 400+M+M
370 GOSUB 400+Z+Z
375 RETURN
400 PR 0;
401 RETURN
402 PR 1;
403 RETURN
404 PR 2;
405 RETURN
406 PR 3;
407 RETURN
408 PR 4;
409 RETURN
410 PR 5;
411 RETURN
412 PR 6;
413 RETURN
414 PR 7;
415 RETURN
416 PR 8;
417 RETURN
418 PR 9;
419 RETURN
420 PR "A";
421 RETURN
422 PR "B";
423 RETURN
424 PR "C";
425 RETURN
426 PR "D";
427 RETURN
428 PR "E";
429 RETURN
430 PR "F";
431 RETURN
499 REM SUB TO PRINT 4 ADDRESSES
500 PR P,
505 Z=USR(S+20,T)
510 GOSUB 350
515 PR "      ";
520 Z=USR(S+20,U)
525 GOSUB 350
530 PR "      ";
535 Z=USR(S+20,V)
540 GOSUB 350
545 PR "      ";
550 Z=USR(S+20,W)
555 GOSUB 350
560 PR
565 X=X+1
570 RETURN
800 REM SUB ADDRESS(HEX TO DEC)
802 N=0
805 X=1
807 PR "INPUT ADDRESS"
810 INPUT R
815 GOSUB 900
820 IF X=4 RETURN
825 X=X+1
830 GOTO 810
850 REM SUB DATA(HEX TO DEC)
852 N=0
855 X=1
857 PR "INPUT DATA"
860 INPUT R
865 GOSUB 900
870 IF X=2 RETURN
875 X=X+1
880 GOTO 860
900 REM HEX TO DECIMAL SUB
905 IF R>999 THEN N=N*16
910 IF R>99 THEN N=N*16
915 IF R>9 THEN N=N*16
920 IF R>0 GOTO 990
925 IF R<0 THEN R=-R
930 N=N*16+R
935 RETURN
990 R=R+R/1000*1536+R/100*96+R/10*6
995 GOTO 925
1000 END
```

---------LISTING #2---------

```
                        .OR 0200        START ADDRESS
                   PHY. .DL 0001        LABELS
                   EMT. .DL 0002
                   INT. .DL 0003
                   CRIT .DL 0004
:
0200  A9 00              LDA 00         START
0202  85 04              STA *CRIT
0204  F8            STAR SED
0205  18                 CLC
0206  A5 01              LDA *PHY.      INCREMENT PHY
0208  C9 23              CMP 23
020A  F0 07              BEQ SET1
020C  69 01              ADC 01
020E  85 01              STA *PHY.
0210  4C 17 02           JMP EMOT
0213  A9 01         SET1 LDA 01
0215  85 01              STA *PHY.
0217  A5 02         EMOT LDA *EMT.      INCREMENT EMT
0219  C9 28              CMP 28
021B  F0 07              BEQ SET2
021D  69 01              ADC 01
021F  85 02              STA *EMT.
0221  4C 28 02           JMP INTL
0224  A9 01         SET2 LDA 01
0226  85 02              STA *EMT.
0228  A5 03         INTL LDA *INT.      INCREMENT INT
022A  C9 33              CMP 33
022C  F0 07              BEQ SET3
022E  69 01              ADC 01
0230  85 03              STA *INT.
0232  4C 39 02           JMP PCRT
0235  A9 01         SET3 LDA 01
0237  85 03              STA *INT.
0239  A5 01         PCRT LDA *PHY.      PHY CRITICAL?
023B  C9 01              CMP 01
023D  F0 19              BEQ LOP1
023F  C9 12              CMP 12
0241  F0 15              BEQ LOP1
0243  A5 02         ECRT LDA *EMT.      EMT CRITICAL?
0245  C9 01              CMP 01
0247  F0 14              BEQ LOP2
0249  C9 15              CMP 15
024B  F0 10              BEQ LOP2
024D  A5 03         ICRT LDA *INT.      INT CRITICAL?
024F  C9 01              CMP 01
0251  F0 0F              BEQ LOP3
0253  C9 17              CMP 17
0255  F0 0B              BEQ LOP3
0257  60            EXIT RTS
0258  E6 04         LOP1 INC *CRIT     INCREMENT CRIT
025A  4C 43 02           JMP ECRT
025D  E6 04         LOP2 INC *CRIT
025F  4C 4D 02           JMP ICRT
0262  E6 04         LOP3 INC *CRIT
0264  4C 57 02           JMP EXIT
                   END. .EN
```

SYMBOL TABLE

PHY.	0001
EMT.	0002
INT.	0003
CRIT	0004
STAR	0204
SET1	0213
EMOT	0217
SET2	0224
INTL	0228
SET3	0235
PCRT	0239
ECRT	0243
ICRT	024D
EXIT	0257
LOP1	0258
LOP2	025D
LOP3	0262
END.	0267

---------SAMPLE #1---------

```
:RUN

COMMAND MODE----SELECT ONE
   0. DEFINE SUBROUTINE ADDRESS
   1. DEFINE PRINT ADDRESSES
   2. DEFINE PRINT MODE
   3. PRESET DATA
   4. PRINT LIMIT
   5. RUN PROGRAM
   6. EXIT PROGRAM
   7. SEE COMMAND OPTIONS
? 0

INPUT ADDRESS
? 0,2,0,0

---COMMAND MODE---
? 1

POSITION #1--INPUT ADDRESS
? 0,0,0,1

POSITION #2--INPUT ADDRESS
? 0,0,0,2

POSITION #3--INPUT ADDRESS
? 0,0,0,3

POSITION #4--INPUT ADDRESS
? 0,0,0,4

---COMMAND MODE---
? 5
```

LOOP	-1-	-2-	-3-	-4-
1	02	02	12	00
2	03	03	13	00
3	04	04	14	00
4	05	05	15	00
5	06	06	16	00
6	07	07	17	01
7	08	08	18	00
8	09	09	19	00
9	10	10	20	00
10	11	11	21	00

---------SAMPLE #2---------

```
---COMMAND MODE---
? 3

INPUT ADDRESS
? 0,0,0,1

INPUT DATA
? 2,0

ANY MORE TO PRESET?(1=N 2=Y)
? 2

INPUT ADDRESS
? 0,0,0,2

INPUT DATA
? 2,0

ANY MORE TO PRESET?(1=N 2=Y)
? 2

INPUT ADDRESS
? 0,0,0,3

INPUT DATA
? 2,0

ANY MORE TO PRESET?(1=N 2=Y)
? 1

---COMMAND MODE---
? 4

INPUT PRINT LIMIT
? 15

---COMMAND MODE---
? 5
```

LOOP	-1-	-2-	-3-	-4-
1	21	21	21	00
2	22	22	22	00
3	23	23	23	00
4	01	24	24	01
5	02	25	25	00
6	03	26	26	00
7	04	27	27	00
8	05	28	28	00
9	06	01	29	01
10	07	02	30	00
11	08	03	31	00
12	09	04	32	00
13	10	05	33	00
14	11	06	01	01
15	12	07	02	01

```
---------SAMPLE #3---------
---COMMAND MODE---
? 4

INPUT PRINT LIMIT
? 4

---COMMAND MODE---
? 2

PRINT MODE---SELECT ONE
   1. ALL LOOPS
   2. DEFINE NUMBER OF LOOPS
   3. CONDITIONAL PRINT
? 2

INPUT LOOP INCREMENT
? 23

---COMMAND MODE---
? 5

LOOP    -1-     -2-     -3-     -4-

23      12      02      25      01
46      12      25      15      01
50
69      12      20      05      01
92      12      15      28      02
```

```
---------SAMPLE #4---------
---COMMAND MODE---
? 2

PRINT MODE---SELECT ONE
   1. ALL LOOPS
   2. DEFINE NUMBER OF LOOPS
   3. CONDITIONAL PRINT
? 3

INPUT ADDRESS
? 0,0,0,4

INPUT DATA
? 0,2

---COMMAND MODE---
? 5

LOOP    -1-     -2-     -3-     -4-

50
100
138     12      13      01      02
150
154     05      01      17      02
196     01      15      26      02
200
250
253     12      16      17      02
```

THE OSI FLASHER:
BASIC-MACHINE CODE INTERFACING

Robert E. Jones
Handley High School
West Point St.
Roanoke, AL 36274

The following program is an example of how a machine language program for the 6502 microprocessor may be loaded from BASIC, executed, and then control may be returned to BASIC (and back again and again, as in this case.) I wrote the program to use in my job as a science teacher at Handley High School in Roanoke, Alabama, where we have two 6502 based microcomputers to use in teaching programming and solving problems of a repetitive nature in chemistry and physics. This program is set up to be run on our OHIO SCIENTIFIC CHALLENGER II.

Our CHALLENGER was originally a MODEL 65V-4K with a total of 12K of RAM. It has been updated with the new MODEL 500 CPU board with OSI MICROSOFT 8K BASIC in ROM. We also use a COMMODORE PET with 8K of RAM for programs which need graphics.

The program may be run on any OSI challenger with a video board set up to start at screen memory location 53312 (base 10) or DOOO (hex). Our video board is the old 440 BOARD with only four pages of screen memory. The new MODEL 540 video

```
10  FOR Y = 1 TO 32 : PRINT : NEXT Y
20  PRINT "INPUT THE DELAY CONSTANT."
30  PRINT "USE A LOW NUMBER FOR A"
40  PRINT "FAST FLASH RATE ( <.5)."
50  INPUT T
60  FOR P = 4096 TO 4130
70  READ C : POKE P,C
80  NEXT P
90  POKE 11,0 : POKE 12,16
100 FOR X = 53200 TO 54380
110 POKE X,INT(255*RND(8))
120 NEXT X
130 FOR D = 1 TO 100*T
140 NEXT D
150 X = USR (X)
160 GOTO 130
170 DATA 160,4,162,0,189,0,208
180 DATA 105,1,157,0,208,232,208
190 DATA 245,238,6,16,238,11,16
200 DATA 136,208,236,169,208,141,6
210 DATA 16,141,11,16,96,0,16
```

```
                    OSI FLASHER

                    BY ROBERT E. JONES
                    FEBRUARY 1979

1000                        ORG   $1000

1000 A0 04      START   LDYIM $04     LOAD INDEX Y WITH 4
1002 A2 00              LDXIM $00     LOAD INDEX X WITH 0
1004 BD 00 D0   LOOP    LDAX  $D000   LOAD A WITH CONTENTS OF D000 + X
1007 69 01              ADCIM $01     ADD 1
1009 9D 00 D0           STAX  $D000   STORE AT D000 + X
100C E8                 INX           BUMP POINTER/COUNTER
100D D0 F5              BNE   LOOP    BRANCH IF NOT ZERO
100F EE 06 10           INC   $1006   INCREMENT ADDRESSES
1012 EE 0B 10           INC   $100B
1015 88                 DEY           DECREMENT INDEX Y
1016 D0 EC              BNE   LOOP    LOOP IF NOT ZERO
1018 A9 D0              LDAIM $D0
101A 8D 06 10           STA   $1006
101D 8D 0B 10           STA   $100B
1020 4C 00 10           JMP   START   CONTINUE RUNNING
```

WARNING: Set the BASIC LOMEM pointer to some address above this machine language code before running BASIC, or you will destroy the code.

board-based Challengers may use this program to occupy all eight pages of video memory if a change is made on line 170. The number to be changed is the second number of the DATA statement, the number which tells the program how many pages of screen memory to use. For 540-based systems the new version should be as follows:

170 data 160,8,162,0,189,0,208

The reason for the ease of change is that the starting locations for the screen memory on both the 440 and 540 boards is the same, D000 (hex). The latter version with the provision for eight pages of video display will work on either type of board, but it seems tedious to me to poke numbers into screen memory locations not visible on my 440-based machine.

LINE 10	CLEARS THE SCREEN
LINES 20-50	GIVE INSTRUCTIONS AND INPUT THE DELAY FACTOR. THE LARGER THE DELAY FACTOR, THE SLOWER THE FLASH RATE.
LINES 60-80	READ THE MACHINE CODE PROGRAM AND STORE IT IN MEMORY LOCATIONS 4096 TO 4130 (DECIMAL) OR 1000 TO 1022 (HEX).
LINE 90	POINTS TO THE START OF THE USR ROUTINE - WHERE TO JUMP TO WHEN EXITING FROM BASIC.
LINES 100-120	CREATE A SCREEN FULL OF RANDOM CHARACTERS
LINES 130-140	DELAY ROUTINE TO ALLOW THE SCREEN TO REMAIN AS IS FOR A TIME DEPENDING ON THE SIZE OF THE DELAY FACTOR BEFORE RETURNING TO THE MACHINE CODE PROGRAM.
LINE 150	CAUSES AN EXIT FROM BASIC TO THE MACHINE CODE PROGRAM
LINE 160	SENDS THE PROGRAM BACK TO THE DELAY ROUTINE WHILE IN BASIC.
LINES 170-210	DATA STATEMENTS FOR THE MACHINE CODE PROGRAM

Mike Rowe
P.O. Box 3
So. Chelmsford, MA 01824

Name: Bridge Challenger
System: PET or Apple II
Memory: 8K PET or 16K Apple II
Language: Not specified
Hardware: Not specified
Description: Bridge Challenger lets you and the dummy play four person Contract Bridge against the computer. The program will deal hands at random or according to your criterion for high card points, and you can save hands on cassette and reload them for later play. You can review tricks, rotate hands East-West, shuffle only the defense hands, or replay hands when the cards are known.
Copies: Not specified
Price: $14.95
Includes: Not specified
Author: Not specified
Available from:
Personal Software
P.O. Box 136
Cambridge, MA 02138
617/783-0694

Name: CURSOR - Programs for PET Computers
System: PET
Memory: 8K
Language: BASIC and Assembly Language
Hardware: Standard PET
Description: CURSOR is a cassette magazine with proven programs written just for the 8K PET. Each month the subscriber receives a C-30 cassette with five or more high quality programs for the PET. People can't read this "magnetic magazine", but the PET can! The CURSOR staff includes professional programmers who design and write many of the programs. They also carefully edit programs which are purchased from individual authors.
Copies: Not specified
Price: $24 for 12 monthly issues
Includes: Cassette
Authors: Many and varied
Available from:
Ron Jeffries, Publisher
CURSOR
P.O. Box 550
Goleta, CA 93017
805/967-0905

Name: PET Schematics and PET ROM Routines
System: PET
Memory: None
Language: None
Hardware: None
Description: PET Schmatics is a very complete set of accurately and painstakingly drawn schematics about your PET. It includes a 24" x 30" CPU board, plus oversized drawings of the Video Monitor and Tape Recorder, plus complete Parts layout - all the things you hoped to get from Commodore, but didn't!
PET ROM Routines are complete assembly listings of all 7 ROMs, plus identified subroutine entry points.
Copies: Not specified.
Price: PET Schematics - $34.95
PET ROM Routines - $19.95
Available from:
PET-SHACK Software House
Marketing and Research Co.
P.O. Box 966
Mishawaka, IN 46544

Name: S-C Assembler II
System: Apple II
Memory: 8K
Language: Assembly language
Hardware: Apple II, optional printer
Description: Combined text editor and assembler carefully integrated with the Apple II ROM-based routines. Editor inclues full Apple II screen editing, BASIC-like line-number editing, tab stops, and renumbering. LOAD, SAVE, and APPEND commands for cassette storage. Standard Apple II syntax for opcodes and address modes. Labels (1 to 4 characters), arithmetic expressions, and comments. English language error messages. Monitor commands directly available within assembler. Speed and suspension control over listing and assembly.
Copies: Just released, over 100 sold.
Price: $20.00 (Texas residents add 5% tax)
Includes: Cassette in Apple II format and a 28 page reference manual.
Author: Bob Sander-Cederlof
Available from:
S-C Software
P.O. Box 5537
Richardson, TX 75080

Name: PL/65 or CSL/65
System: SYSTEM 65 or PDP 11
Memory: 16K bytes RAM
Language: Machine language.
Hardware: Rockwell SYSTEM 65
Description: A high-level language resembling PL/1 and ALGOL is now available to designers developing programs for the 6500 microprocessor family using either the SYSTEM 65 development system of the PDP 11 computer. PL/65 is considerably easier to use than assembly language or object code. The PL/65 compiler outputs source code to the SYSTEM 65's resident assembler. This permits enhancing or debugging at the assembler level before object code is generated. In addition, PL/65 statements may be mixed with assembly language instructions for timing or code optimization.
Copies: Not specified.
Price: Not specified from Rockwell.
$500 from COMPAS.
Includes: Minifloppy diskette.
Authors: Not specified.
Available from:
Electronic Devices Division
Rockwell Internationsal
P.O. Box 3669
Anaheim, CA 92803
714/632-2321 (Leo Scanlon)
213/386-8776 (Dan Schlosky)

COMPAS - Computer Applications Corp.
413 Kellogg
P.O. Box 687
Ames, IA 50010
515/232-8181 (Michael R. Corder)

NOTE: Since some of these Software Catalog listings appeared as long ago as Oct. 1978, the reader is advised to check with the vendor to determine current availability, price, etc.

Name: PRO-CAL I
System: PET
Memory: Not specified.
Language: BASIC and machine language.
Hardware: Not specified.
Description: A reverse polish scientific calculator program, ideally suited for scientific and educational applications. Supports single key execution of more than 50 forward and inverse arithmetic, algebraic, trigonometric and exponential functions. It implements calculations in binary, octal, decimal, and hexidecimal modes with single keystroke conversion between modes and simultaneous decimal equivalen display.It also allows the recording and playback of calculator programs on cassette tape permitting the use of most calculator software already in existance up to a limit of 255 steps.
Copies: Not specified.
Price: $26.00 domestic, $28.00 foreign.
Includes: Software on cassette and an operating manual.
Authors: Not specified.
Available from:
Applications Research Co.
13460 Robleda Road
Los Altos Hills, CA 94022

Name: MICROCHESS
Systems: PET and Apple II
Memory: PET - 8K/Apple II 16K
Language: 6502 Machine Language
Hardware: Standard PET or Apple II
Description: MICROCHESS is the culmination of two years of chessplaying program development by Peter Jennings, author of the famous 1K byte chess program for the KIM-1. MICROCHESS offers eight levels of play to suit everyone from the beginner learning chess to the serious player. It examines positions as many as 6 moves ahead, and includes a chess clock for tournament play. Every move is checked for legality and the current position is display on a graphic chessboard. You can play White or Black, set up and play from special board positions, or even watch the computer play against itself.
Copies: Not specified.
Price: $19.95
Includes: Not specified.
Author: Peter Jennings
Available from:
Personal Software
P.O. Box 136
Cambridge, MA 02138
617/783-0694

Name: Financial Software
System: Apple II (easily modified for PET)
Language: Applesoft II
Hardware: Apple II
Description: Sophisticated financial programs used to aid in investment analysis. The following programs are currently available: Black-Scholes Option Analysis, Security Analysis using the Capital Asset Pricing Model, Bond Pricing I and II, Cash Flow and Present Value Analysis I and II, Stock Valuation, Rates of Return, Calculations and Mortgage Analysis.
Copies: Just released.
Price: $15.00 each or $50.00 for all 9 programs
Includes: Cassette, annotated source listings, operating and modifying instructions, sample runs and backgroud information.
Author: Eric Rosenfeld
Available from:
Eric Rosenfeld
70 Lancaster Road
Arlington, MA 02174

Name: Apple II BASEBALL
System: Apple II
Memory: 16K or more
Language: Integer BASIC
Hardware: Standard Apple II
Description: An interactive baseball game that uses color graphics extensively. You can play a 7 or 9 inning game with a friend, (it will handle extra innings), or play alone against the computer. Has sound effects with men running bases. Keeps track of team runs, hits, innings, balls and strikes, outs, batter-up and uses paddle input to interact with the game. Uses every available byte of memory.
Copies: Just released.
(Dealers inquiries invited)
Price: $12.50
Includes; Game Cassette, User Bookelt with complete BASIC listing.
Authors: Pat Chirichella and Annette Nappi
Available from:
Pat Chirichella
506 Fairview Avenue
Ridgewood, NY 11237

Name: DDT-65 Dynamic Debugging Tool
System: Any 6502 based system
Memory: 3K RAM/1K RAM for loader
Language: Machine Language
Hardware: 32 char/line terminal
Description: DDT-65 is an advanced debugger that allows easy assembly and disassembly in 650X mnemonics. Software single-stepping and automatic breakpoint insertion/deletion allow debuffing of code even in PROM. DDT-65 comes in a relocatable form on tape for loading into any memory or for PROM programming.
Copies: 11+
Price: $25.00
Include: 10 page manual, relocating tape cassette.
Ordering Info: KIM format cassette - K
Kansas City at 300 baud for OSI - O
Kansas City at 300 baud for TIM/JOLT - T
Author: Rich Challen
Available from:
Rich Challen
939 Indian Ridge Drive
Lynchburg, VA 24502

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

Name: **Text Editor/Word Processor**
System: **Apple II**
Memory: **24K for cassette, 32K for Disk II**
Language: **Applesoft II**
Hardware: **Apple II, cassette tape recorder or Disk II and printer**
Description: Uses any width line, features upper and lower case using inverse video, justification by adding blanks, user set and cleared tabs in any column, automatically renumbers lines on insertion or deletion, usable with any printer interface by extremely slight program modifcation.
Copies: **100+**
Price: **$50. for cassette version, $60 for Disk version**
Includes: cassette or diskette and instructions. Source listing available by sending SASE with serial number
Author: **Craig Vaughn**
Available from: Local Apple dealers or:
Peripherals Unlimited
6012 Warwood Road
Lakewood, CA 90713

Name: **Mailing Label Package**
System: **Apple II**
Memory: **At least 32K**
Language: **Applesoft II**
Hardware: **Apple II, Disk II, and printer**
Description: Stores 3-line or 4-line addresses (may be mixed) plus phone # and a 15-character code field, any one record may be accessed by name or phone #, prints in zip code order, will print all records or select by code field with wild card, any number of labels horizontally, user formats spacing, may be used with any printer interface with very slight program modification. Five hundred records maximum on one diskette with 48K.
Copies: **20**
Price: **$40**
Includes: Diskette and instructions. Source listing available by sending SASE with serial number.
Author: **Claudia Vaughn**
Available from: Local Apple dealers or:
Peripherals Unlimited
6012 Warwood Road
Lakewood, CA 90713

Name: **APPLE PILOT**
System: **Apple II**
Memory: **16K tape I/O, 32K Disk I/O**
Language: **Interpreter in Applesoft II**
Hardware: **Apple II**
Description: A language to write games and school lessons with. Only 8 commands to learn plus special Apple graphics and tone commands.
Copies in circulation: **10**
Price: **$20.** Add $5 for a diskette.
Includes: Tape and manual and 1 year updates.
Author: **Earl Keyser**
Available from:
The Pilot Exchange
22 Clover Lane
Mason City, LA 50401

Name: **Programs for Indoor Advertising Applications**
System: **Apple II**
Memory: **16K**
Language: **Integer BASIC and Machine Language**
Hardware: **Standard Apple II**
Description: This Program allows the Apple to be used as an automated Advertising machine for stores, trade shows, etc.

HI-RES ALPHANUMERIC MESSAGES: 28 Characters per line, 4 lines, 3 pages of text. Features a right-side 'word-rap' plus instant 'page desolve', as one page ends and the next begins. Characters are crisp and can be Lavender or Green on a Black Background. They 'puff' on at reading speed.

GIANT-LETTER SEQUENCES: Brilliantly-colored letters, of full screen height appear one-at-a-time, in sequence, to spell out messages. The color of Successive Words progresses through the Apple rainbow. A running summary of letters appears in the bottom four screen lines, as the giant letters are presented.

THE SCROLLING WONDER: Allows user to enter up to four brief messages. They appear in Apple upper case by 'popping' onto the screen from below. Messages enter in random sequence, with random space between them. They have random horizontal placement and a random 50% sample of the messages 'flash'. A multiple-rainbow grand finale ends the program.

Copies: **All just released**
Prices: SCROLLING WONDER **$8.00**
GIANT-LETTER SEQUENCES **$8.00**
HI-RES ALPHANUMERIC MSG **$15.00**
ALL THREE PROGRAMS **$25.00**
Includes: Cassette only, with verbal instructions on reverse side of cassette and written instructions on screen.
Author: **Howard Rothman**
Available from:
Connecticut Information Systems Co.
218 Huntington Road
Bridgeport, CT 06608
203/579-0472

Name: **Hangman**
System: **Apple II**
Memory: **20K minimum**
Language: **BASIC**
Hardware: **Apple II, Disk II**
Description: This program is the old traditional Hangman we used to play with pencil and paper except that the computer will choose the word for you to guess. The disk comes with over 350 words and has routines accessed with 'ESC' to add or change words. Gallows is in lores and neck stretches when floor drops.
Copies: **Aprox. 25**
Price: **$14.00** post paid. Calif. residents add sales tax
Includes: Disk with program and over 350 words.
Order Info: Master Charge and Visa accepted.
Author: **Loy Spurlock**
Available from:
Computer Forum Company
14052 E. Firestone Blvd.
Santa Fe Springs, CA 90670

Name: **Feet and Inches Calculator**
System: **Apple II**
Memory: **16K**
Language: **Applesoft ROM**
Hardware: **Applesoft ROM**
Description: This program does calculations based on entries made in feet and inches. Functions include addition, subtraction, division, multiplication, roots, powers and decimal equivalents. Operating screen consists of three windows: one for entries, one lists functions, and the third reproduces the problem after entry. Performs calculations to 1/64". Has memory which allows recall of last answer for next problem.
Copies: **Just released**
Price: **$10.00**
Includes: Cassette tape
Author: **Dick Dickinson**
Available from:
Dick Dickinson
5400 Western Hills Drive
Austin, TX 78731

Name: **BLOCKADE**
Systems: **Challenger IIP**
Memory Required: **4K**
Language: **BASIC and assembly**
Hardware Required: **Challenger II or III**
Description: Two players are needed to play this challenging game in which the object is to block out your opponent before he blocks you out! Each play has four keys for NESW direction, which enable you to construct a wall, trying to block out the other player. The first person to run into the wall loses. Programmed for large characters, or small. Uses Assembly for fast clearing of the screen and printing of characters. Complete with scoring.
Copies: **Lots!**
Price: **$8.00** for listing, cassette, and instructions.
$4.00 for listing and instructions only.
Includes: Cassette at 300 Baud. ($8).
Author: **Bill Langford**
Available from:
Bill Langford
3823 Malec Circle
Sarasota, Fla. 33583

Name: **OSI Games**
System: **OSI Superboard II/Challenger 1P**
Memory: Not specified
Language: Not specified
Hardware: Not specified
Description: **Dodgem** - use strategy to get your pieces off the opposite side of the board (1 or 2 players). **Tank Attack** - seek and destroy enemy guns hidden among houses and trees before they get you (1 player). **Free-for-all** - airplane, destroyer, and submarine vie for each other (1 or 2 players). **Hidden Maze** - find your way through an invisible maze with one-way gates (1 or 2 players).
Copies: Not specified
Price: **$7.95** († 75 cents postage)
Includes: Tape cassette, instruction booklet.
Author: Not specified
Available from: A large number of dealers or:
Creative Computing Software
P.O. Box 789-M
Morristown, NJ 07960
201/540-0445

Name: **3D Graphics**
System: **Apple II**
Memory: **16K**
Language: **Floating Point BASIC**
Hardware: **Apple II** (Applesoft ROM for Load and Go option)
Description: Accurate 3D to 2D wire frame perspective transformations of your data bases. The standard software package contains the BASIC listing for transformation of 3D line endpoints (X,Y,Z coordinates) to perspective drawing endpoints in two dimensions (X,Y coordinates) for high-resolution plotting. User has control over location in space, direction of view, and viewing window (telephoto or wide angle). User must be able to run floating point BASIC and hi-res graphics simultaneously. Optional Load and Go version is specifically for Applesoft ROM and includes a sample data base and output-plotting interface. It is truly Load and Go.
Copies: **Over 200 sold**
Price: **$22** ($26 with Load and Go option)
Includes: 60 page manual and listing (Applesoft II cassette with Load and Go option)
Author: **Bruce Artwick** (option by **Jim Harter**)
Available from:
SubLOGIC
P.O. Box V
Savoy, IL 61874
217/367-0299

Name: **Program Catalog**
System: **Apple II**
Memory: **24K minimum**
Language: **BASIC**
Hardware: **Apple II, Disk II**
Description: This program will catalog all your disk programs by category on one disk. It will keep track of all your programs and which disks they are on as well as keeping notes about the program so you can be sure of the program before you hit the proper key to have this program load and run the program you want. It also contains numerous routines to manipulate the information.
Copies: **New, just released.**
Price: **$19.00** post paid. Calif. residents add sales tax.
Includes: Program on disk, documentation
Order Info: Master Charge and Visa accepted.
Author: **Loy Spurlock**
Available from:
Computer Forum Company
14052 E. Firestone Blvd.
Santa Fe Springs, CA 90670

Editor's Note: The **MICRO Software Catalog** was the most mentioned article in our recent reader survey. If you have software you would like to bring to the attention of the **MICRO** readers, simply type it up in the proper format and send it in. Please adhere to the format as strictly as possible, including UPPER and lower case, titles, and so forth. Since this material will be typeset someone has to get it into proper form. If you submit it in proper form, you increase your chances for early inclusion in **MICRO**. There is no charge for appearing in this catalog.

We are happy to see some programs for the OSI systems appearing.

Name: **MAXIT!**
System: **PET**
Memory: **8K**
Language: **BASIC**
Hardware: **Standard**
Description: A challenging number game played between two persons or versus the PET. From an 8 × 8 board players alternatively move horizontally and vertically trying to maximize their score and minimize their opponents. An exciting, engrossing game, that bears returning to multiple times. Suitable for young and old alike. Excellent graphics.
Copies: **50 plus**
Price: **$4.95** plus 32¢ tax for CA residents, pp.
Includes: Cassette and 2 page printed instructions.
Author: **Harry J. Saal**
Available from:
Harry J. Saal
810 Garland Drive
Palo Alto, CA 94303

Name: **6502 Tiny Editor - Assembler**
System: **Any 6502 based system.**
Memory **Program takes 1K, 4K** recommended for source and object code and label table.
Language: **Machine Language**
Hardware: **ASCII Keyboard** and **CRT** display.
Description: A single pass assembler, closely follows MOS Mnemonics, and is extremely memory efficient. The editor is designed to be easily extended by the user. Editor commands include: Find line, delete line, insert line, list source, list symbolic labels, define label, and set origin. A single pass assembler allows the object code to overwrite the source code - larger source programs can be assembled in a given memory size.
Copies: **Just released:**
Price: **$19.95** (KIM-1 Hypertape cassette: $3.00 extra)
Include: User manual and complete source and object listing, fully commented, with modification instructions.
Author: **Michael Allen**
Available from:
Michael Allen
6025 Kimbark
Chicago, IL. 60637

Name: **6502 ROBOT**
System: **Any 6502 based system**
Memory: **1.5K**
Language: **Machine language**
Hardware: **ASCII Keyboard** and **CRT** display, or "turtle", or plotter.
Description: ROBOT is an interactive programming language for the control of robots, such as "turtle", plotter or CRT cursor. ROBOT's command processing module is designed to allow the user to design his own language of personalized commands and command subroutines to suit his particular application. The version offered here includes a command set and subroutine package for the control of a CRT robot.
Copies: **Just released.**
Price: **$5.00** (KIM-1 Hypertape cassette: **$3.00** extra)
Include: user manual, complete and fully commented source and object listing, instructions for adapting, modifying, and using the command processing module for other applications.
Author: **Michael Allen**
Available from:
Michael Allen
6025 Kimbark
Chicago, IL 60637

Name: **OSI Games**
System: **Challenger**
Memory: **4K 8K**
Language: **Basic and Assembly**
Hardware: **Challenger**
Description: The game programs are written for the challenger with the 440 video display and ASCII keyboard. Most of these will run on the 2p and 1p. Games such as Bomber and Klingon are written with simulated animation and Klingon also will support sound with PIA port and tone oscillator. We also have lunar lander; Battleship; and others.
Copies: **Just released**
Price: **$8.00** for listing and instructions and 300 baud cassette
Author: **William L. Taylor**
Available from:
William L. Taylor
264 Flora Rd.
Leavittsburg, Ohio 44430

Name: **LINK**
System: **PET**
Memory: **Any amount**
Language: **Assembly**
Hardware: **Standard PET**
Description: This program will allow the user to link exclusively numbered BASIC programs in memory. This allows the programmer to develop complex programs as sub modules and then merge them together into the final functioning unit. A great time saver as the programmer can develop a library of subroutines which can be merged virtually at any time with the program which he is developing. With complete instructions on use.
Copies: **Just released**
price: **$12.95** ppd, Michigan residents add 4 % sales tax.
Includes: **Cassette and instructions**
Author: **G. Salked**
Order Info.: Master Charge and Visa accepted.
Available from:
Your local PET dealer or
Dr. Daley
425 Grove Ave.
Berrien Springs, MI 49103
616-471-5514

Name: **PILOT**
System: **PET**
Memory: **8K minimum**
Language: **BASIC**
Hardware: **Standard PET**
Description: A simple to use, easy to learn programming language. This is especially suited for use by children. Only 10 commands to learn with no complicated syntax plus special cursor and graphics control commands.
Copies: **25**
Price: **$12.95** ppd, Michigan residents add 4% sales tax.
Includes: cassette and users manual.
Author: **R.F. DAley**
Orther Info.: **Master Charge and Visa accepted**
Available from:
Your local PET dealer or
Dr. Daley
425 Grove Ave.
Berrien Springs, MI 49103
616-471-5514

Name: Slow-Scan Television Package
System: Apple II
Memory: 16K (min)
Language: Machine Language
Hardware: Standard Apple II
Description: This software system allows the Apple II to send and receive U.S. amateur standard slow-scan T.V. pictures (120 line-15 Hz) via any ham radio SSB transceiver. A real-time display of the received picture in high-resolution graphics is accomplished with a sophisticated image processing algorithm. Low-resolution images for transmission are prepared with a large-character display editor as well as a drawing editor. All modulation and demodulation of the audio FM subcarrier is performed by the software — replacing hundreds of dollars of hardware required by other SSTV systems. Comes on cassette with 8 mins. of test pictures.
Copies sold: about 100
Price: $20.
Includes: Cassette tape and 5 pages of documentation.
Author: Chris H. Galfo — WB4JMD
Available from:
C.H. Galfo
602 Orange St
Charlottesville, VA 22901

Name: S-C Assembler II (disk version)
System: Apple II with at least one disk
Memory: 24K or more
Hardware: Apple II, Disk II, optional printer
Description: Disk version of the popular S-C Assembler for the Apple II. Combines a text editor and an assembler in one memory resident package of 3072 bytes (1000-1BFE). Carefully integrated with the Apple II ROM-resident routines, and with Apple DOS. Editor includes full screen-editing, BASIC-like line number editing, tab stops, and renumbering. LOAD and SAVE commands for storage of source programs on disk files or cassette. JOIN command for appending two source programs from cassette. Standard Apple II syntax for opcodes and address modes. Labels (up to 6 characters), arithmetic expressions, comments in a liberated line format. English language error messages (not coded numbers). DOS and Apple Monitor commands directly available within the assembler. Speed and suspension control over listing and assembly. Includes printer deiver for Practical Automation printer, with instructions for modification to any other printer. (Cassette version is still available: it has fixed line format and labels up to four characters.)
Copies: Over 200 of cassette version, over 25 of disk version.
Price: $35 for disk version, $25 for cassette version (Texas residents add 5% sales tax)
Includes: 32-page reference manual, disk with assembler, Master. Create, RAWDOS, and two sample source programs.
Author: Bob Sander-Cederlof
Available from:
S-C SOFTWARE
P.O. Box 5537
Richardson, TX 75080

Name: PRO-CAL-I
System: Commodore PET
Memory: 8K
Language: Microsoft BASIC
Hardware: PET
Description: PRO-CAL-I is a reverse polish programmable scientific calculator program ideally suited to scientific and educational applications. It combines the best features of the PET with those of hand-held calculators such as the HP 97 and the TI "Programmer". It supports single key execution of more than 50 functions and implements calculations in binary, octal, decimal, and hexidecimal number systems. The program displays 10 memory registers, 5 stack registers, and a record of the 14 most current operations.
Copes: 40
Price: $26.00 for software on cassette and an operating manual.
Author: Robert M. Munoz
Available from:
APPLICATIONS RESEARCH CO.
13460 Robleda Rd.
Los Altos Hills, CA 94022

Name: EINANCIAL ANALYSIS: A Tutorial
System: APPLE II and PET
Memory: 16K
Language: Basic
Hardware: APPLE II with cassette recorder, or a PET (8K)
Description: An interactive learning cassette with chapters on Risk, Short-term and Intermediate-term Financing, Einancial Statements, and Key Business Ratios. The user is then put into the position of having to use these concepts by playing the Meany Manufacturing Business Game.
Copies: Hundreds available
Price: Sugg. Retail: $16.50
Includes: Tape cassette and informative booklet
Author: Brian Beninger
Available from:
Local APPLE or PET dealers of:
SPEAKEASY SOFTWARE LTD.
P.O. Box 1220
Kemptville, Ont., KOG 1J0

Name: STAT III
System: Commodore PET
Memory: 8K
Language: BASIC
Hardware: Standard PET
Description: STAT III accepts a set of numbers and calculates the following: mean, median, mode, highest number in the data, lowest number in the data, range, variance, standard deviation, average deviation, and sample standard deviation. STAT III can display a bar graph of the users data on the CRT. In addition the user may correct errors in his inputted data before processing.
Copies: Just released
Price: $7.95
Includes: Cassette, source listing (program is self documenting)
Author: Michael J. McCann
Available from:
THE PET PAPER
P.O. Box 43
Audubon, PA 19407

Name: Apple Pi 'Life'
System: Apple II
Memory: 4K
Language: BASIC and assembly
Hardware: Apple II with 2 operable game paddles with switches.
Description: Apple Pi 'Life' allows variable grid sizes from 8X8 up to 40X40 in increments of 1. Paddle 1 is only read when the switch is depressed. Speed is controlled by paddle 0 and can be varied from 550 gpm to 2000 gpm for an 8X8 grid. For a 40X40 grid, speed can be varied from 25 gpm to 140 gpm. The speaker is toggled each time a cell is processed, except at minimum or maximum speed, to give the sounds of 'Life'. The bottom of the grid wraps around to top of grid, and vice-versa. The right of the grid wraps around to left of grid, and vice-versa. There are three tables of pre-defined objects which can be setup on the grid by number and x,y location. A description of the object table structure is given in the documentation. Keyboard controls are: P-pause until next 'P', Z-zero grid and setup objects, O-setup objects on grid, N-new colors, and E-exit program. Any two distinct colors may be used for live and dead cells.
Copies: Nev just released.
Price: $12.00. Texas residents add sales tax.
Includes: Programs, object tables on cassette, documentation.
Order Info: Checks only.
Author: Harry L. Pruetz
Available from:
Microspan Software
2213A Lanier Drive
Austin, TX 78758

Name: Amateur Radio Communications Package
System: Apple II
Memory: 8K (min)
Language: Machine Language and Integer BASIC
Hardware: Apple II and user provided interface
Description: This software package allows the Apple II to communicate in any of three codes: Morse, Baudot, or ASCII, with a minimum amount of external hardware required. Some features include: Variable size text buffer and live keyboard allow preparing text for transmission while receiving or transmitting; 3 field screen display — each field scrolling separately; user defined stored messages are referenced by a keyboard and can be inserted anywhere in the text; automatic 72 character line formatting with word wrap-around; continuously variable code speeds; adaptive Morse receive and lots more! All I/O uses the on-board (game) I/O connector.
Copies sold: over 100
Price: $18.
Includes: Cassette tape and documentation with sample interface.
Author: Chris H. Galfo - WB4JMD
Available from:
C.H. Galfo
602 Orange St.
Charlottesville, VA 22901

Name: TRANSACTIONAL ANALYSIS: An Introduction
System: APPLE II and PET
Memory: 16K
Language: Basic
Hardware: APPLE II with cassette recorder, or a PET (8K)
Description: An introduction to T.A. - a system for understanding human behaviour. Chapters include: You As A Person, Stroking, Transactions, Are You Listening?, the Balancing Game. This interactive learning cassette will help you gain better understanding of why you get along with some people and not with others and may give you a better understanding of yourself!
Copies: Hundreds available
Price: Sugg. Retail: $16.50
Includes: Tape cassette and informative booklet
Author: Joy Karp
Available from:
Local APPLE or PET dealers or:
SPEAKEASY SOFTWARE LTD.
P.O. Box 1220
Kemptville, Ont., KOG 1J0, Canada

Name: DOS TEXT EDITOR
System: APPLE II
Memory: Cassetts-16K, Applesoft Rom-24K, DOS-32K
Language: Applesoft II
Description: EDIT is a program designed to facilitate changes to disk and cassette text files. The program has 24 commands to manipulate files. Included are: INSERT, DELETE, CHANGE, SEARCH, ADD, LIST, TEXT, DISPLAY, PACK, MODE, TAB, CLEAR, APPEND, SAVE, CONCAT, and STRING CHANGE. Commands that operate on blocks of data such as Range DELETE, LIST, SEARCH, and STRING replace are also provided. EDIT may also be used to create Disk files.
Copies: Just released
Price: $16.95 (Add $5 if desired on diskette)
Specify if Applesoft ROM
Includes: Program cassette or diskette, Complete documentation, and users manual.
Author: Robert Stein
Available From:
Services Unique, Inc.
2441 Rolling View Dr.
Dayton, Ohio 45431

Name: REAL-I
System: Commodore PET
Memory: 8K
Language: Microsoft BASIC
Hardware: PET
Description: REAL-I is a real estate investment analysis program which models an investment by computing the cash flow, tax advantage, inflation hedge, internal rate of return, and other quantities as they change over the years under the effects of inflation. It specializes the calculations to the tax position of the investor and helps him to judge the relative merits of various real estate investments opportunities.
Copies: Just released
Price: $29.00 for software on cassette and an operating manual.
Author: Robert M. Munoz
Available from:
APPLICATIONS RESEARCH CO.
13460 Robleda Rd.
Los Altos Hills, CA 94022

Name: **Missile-Anti-Missile**
System: **Apple**
Memory: **16K**
Language: **Apple II Soft**
Description: Simulated missile attack on 3-D Map of USA
Copies: **30**
Price: **$9.95** + **$1.00** postage & handling
Includes: Cassette with instructions
Author: **T. David Moteles & Neil Lipson**
Available from:
Progressive Software
P.O. Box 273
Plymouth Mtg., PA 19462

Name: **DISK DUMP/RESTORE**
System: **Apple II with disk**
Memory: **32K (min)**
Language: **Applesoft II and machine language**
Hardware: **Apple II, Disk II**
Description: A disk-tape utility to dump and restore all Integer, Applesoft II, and Binary programs automatically. The program names, Binary program addresses, and all commands necessary to re-load the programs from tape and restore them again to disk under their original names are stored on tape header file.
Copies: **Just released**
Price: **$8.00**
Includes: Cassette and instructions
Author: **Alan G. Hill**
Available from:
Alan G. Hill
12092 Deerhorn Dr.
Cincinnati, Ohio 45240

Name: **NOT ONE**
System: **KIM**
Memory: **1K**
Language: **Assembly**
Hardware: Bare Kim!
NOT ONE is an exciting, fast moving game of skill, strategy, and change for one to five players (including KIM). The game is designed for use with KIM's onboard display and hex pad.
Besides being an entertainment game, the NOT ONE package was designed to introduce some powerful general-purpose output manipulation subroutines for the KIM's LED display. These include variable-speed, scrolled alpha-numerics!
The manual also discusses LED segment codes in an effort to increase the user's knowledge of the display.
Author: **Steven Wexler**
Price: **$15.00**
Includes: Source listing, manual, and cassette
Available from:
SJW, Inc.
P.O. Box 438
Huntingdon Valley, PA. 19006

The 6502 Program Exch.
2920 Moana
Reno, NV. 89509

Name: **A Forth System**
System: **Apple II**
Memory: **24K or Larger**
Language: **40% ASSEMBLY, 60% Forth**
Hardware: **Disk II**
Description: A unique software package for software buffs and serious programmers who have gotten tired of programming in integer basic and machine language. FORTH is an extensable language, allowing the programmer to "define" new dictionary entrys that use previous entrys. Most of FORTH is written in FORTH. Benchmarks show that FORTH executes 20 times faster than BASIC.
Included in the package are:
1) Powerful screen editor for system development.
2) Decompiler - used to generate to some extent a source listing. It can be used to list our portions of FORTH itself.
3) Utility package - dump, disk maintenance etc. does not use apple II dos.
4) Completely documented using a special disk retreival system. includes some programming examples. Editor, decompiler is available on source.
Copies: **Just Released**
Price: **$39.95** + tax for california residents
Includes: One mini diskette + manual
Author: **John T. Draper**
Available from:
Captain Software
PO Box 575
San Francisco, CA 94101

Name: **Function Graphs and Transformations**
System: **Apple II**
Memory: **16K minimum if Applesoft is in ROM, otherwise 32K minimum**
Language: **Applesoft (floating point Basic)**
Hardware: **No special hardware**
Description: This program uses the Apple II high resolution graphics capabilities to draw detailed graphs of mathematical functions which the user defines in Basic syntax. The graphs appear in a large rectangle whose edges are X and Y scales (with values labeled by up to 6 digits). Graphs can be superimposed, erased, drawn as dashed (rather than solid) curves, and transformed. The transformations available are reflection about an axis, stretching or compressing (change of scale), and sliding (translation). The user can alternate between the graphic display and a text display which lists the available commands and the more recent interactions between user and program. Expected users are engineers, mathmaticians, and researchers in the natural and social sciences; in addition, teachers and students can use the program to approach topics in (for example) algebra, trigonometry, and analytic geometry in a visual, intuitive, and experimental way which complements the traditional, primarily symbolic orientation.
Copies: **Just released**
Price: **$14.95** (Cat. No.: AHE0123)
Includes: cassette tape, 12-page instruction booklet
Author: **Don Stone**
Available from: many computer stores or
Powersoft, Inc.
P.O. Box 157
Pitman, NJ 08071
(609) 589-5500

Name: **6502 VDR**
Systems: Any 6502 with room available at $200 or $DD00
Memory: **½K**
Language: **6502 machine code**
Hardware: **Memory-mapped video board such as Polymorphic Systems VTI, Solid State Music VB-1B, Etc.**
Description: Organizes memory-mapped display for teletype-like use including automatic scrolling, line wrap-around, clear screen commands, etc.
Copies: **30**
Price: **$9.50 plus $1 shipping**
Includes: Operating Manual, detailed configuration information, and complete commented source listing.
Order: Package includes KIM compatable tape cassette with both $200 and $DD00 versions included. Charge cards, phone and mail order accepted.
Available from:
Forethought Products
97070 Dukhobar #D
Eugene, Oregon 97402

Name: **CHEQUE—CHECK™**
System: **PET**
Memory: **8K**
Language: **BASIC, with machine language subroutine**
Hardware: **PET 2001-8 (or 2001-16/32 on special order)**
Description: CHEQUE-CHECK reduces the probability of error in reconciling bank statement and checkbook, even for those experienced in the art. More important it greatly reduces the time required to find and correct an error when one does occur, because it "remembers" individual entries for later review and modification if necessary. Designed and tested for ease of use, CHEQUE-CHECK is suitable for novice or expert, and requires no tape files or knowledge of programming. Reviewed in May 1979 issue of Robert Purser's Reference List of Computer Cassettes.
Copies: **60 sold** in first three months.
Price: **$7.95** (quantity discount available)
Includes: Cassette in Norelco style box, Description and operating instructions, zip-lock protective package.
Designer: **Roy Busdiecker**
Available from: Better computer stores or directly from
Micro Software Systems
P.O. Box 1442
Woodbridge, VA 22193

Name: **Disk Catalog Program**
System: **Apple II**
Memory: **32 K minimum**
Language: **Integer Basic and Machine Language**
Hardware: **Apple II, DISK II**
Description: This program consists of two modules. The first, DCATPRO, is a general purpose data base catalog program for books, records, tapes, programs on diskette, etc. Features include 40 col. records, 5 fields (2 with adjustable length), and super fast machine language sort. The second, GENCPINP, **automatically** processes any set of Apple II diskettes and generates a data base for DCATPRO by reading the D$CATALOG information for each diskettes. Then you know what you have and **where it is,** all without having to type in a lot of data.
Copies: **Over 100 sold**
Price: **$10.00 postpaid**
Includes: Programs on cassette and 5 pages of documentation
Arthur: **George W. Lee**
Available from:
George W. Lee
18003 S. Christina Ave.
Cerritos, California 90701

Name: **Generalized File Management**
System: **APPLE II**
Memory: **16K**
Language: **Integer Basic**
Hardware: **APPLE II, DISK II**
Description: This package allows you to create, update, and print disk files. The names of fields and files, number of fields, individual field lengths, and file size is user defined. You can decide what headings you want to see (if any) when you print or display and record or the entire file. You can use this package to create such files as: Parts lists, phonenos., List of birthdates, name and address, and whatever...
Copies: **Just released**
Price: **$16.50**
Includes: Diskette that contains two programs, some sample file useages (birthdates, parts list), and a user manual.
Author: **Lee Stubbs**
Available from:
Les Stubbs
23725 Oakheath Pl. Harbor City, Ca 90710

Name: **WEAVER**
System: **Apple II**
Memory: **32K**
Language: **Integer Basic**
Hardware: **Disk II**
Description: WEAVER simulates as multi-harness loom with control of warping, hook-up and treadling. Weaving drafts of 40 threads of warp and 40 threads of weft are drawn in 15 colors for patterns requiring up to 24 harnesses. Weaving patterns are saved and called by name from disk storage. The user-interface is designed for easy and efficient use by a weaver. Nine pages of documentation include a glossary of commands which defines the functions of the program and a sample draft with descriptive data entry.
Copies: **New program.**
Price: **$15.00** on cassette tape, **$20.00** on diskette with five sample drafts.
Author: **Bruce Bohannan**
Available from:
Bruce Bohannan
2212 Pine Street
Boulder, CO 80302

Name: **Address and Perpetual Calendar**
System: **APPLE II**
Memory: **32K**
Language: **Applesoft II**
Hardware: **APPLE II w/Disk II**
Description: This program maintains your master address file on disk. User follows a master menu to add or change names, look for specific names or review entire file (or part) name by name. All outputs are formatted. Look and change records with a search function i.e., If you do not remember how to spell a name then enter the number of letters you do know and the program will walk you through all names beginning with what you entered until you find the one you want. A birthday function is included that will search your entire file and list all names, birthday and age for any given month. A special feature loads up a Perpetual Calendar program that will display any month (formatted) between the years 1704 and 2099 and highlights any particular day. Return to address program is optional.
Copies: **Just released.**
Price: **15.00 ppd**
Includes: Disk and instructions
Author: **Edward S. Kleitches**
Available from:
Edward S. Kleitches
7207 Camino Grove
San Antonio, Texas 78227

Name: **DB/65**
System: **ANY 28 or 40 PIN 6500**
Hardware: **Power supply and terminal**
Power Requirements: **5V at 3 AMPS. ‡12, -12 at 20 Milliamps if RS232C terminal used.**
Description: DB/65 is a complete hardware/software debug system for any 6500 system. Command structure is identical to that of the ROCKWELL SYSTEM 65. Hardware breakpoint, scope syne, eight software breakpoints and any number of real-time breakpoints (via the BRK instruction) are supported. Object code and symbol table may be loaded from either serial or parallel port (compatable with SYSTEM 65 printer port). Symbolic disassembly is supported so programmer is always debugging at assembler level. In circuit emulation and 2K RAM are standard. RAM may be added for total of 8K if desired. User NMI and IRQ vectors and supported. System monitor resides in address range $7000 to $7FFF so user program may occupy high memory. 2MHZ option available.
Copies sold: **15**
Price: **$1450**
Includes: Manuals, In circuit emulation, 2K RAM shipping
Developed by: **COMPAS MICROSYSTEMS**
Available from:
COMPAS MICROSYSTEMS
224 SE 16th Street
P.O. Box 687
Ames, IA 50010
515/232-8181

Name: **Home Budget System**
System: **OSI** (Easily modified for PET or Apple II)
Memory: **4K**
Language: **MICROSOFT BASIC**
Hardware: **OSI Challenger IIP**
Description: A computerization of my own proven home budget system evolved over a 7 year period. Consists of interactive programs to add/update accounts, post budget and expenses and analyze status of accounts on detailed and summary basis. 4K RAM handles up to 15 accounts stored on cassette tape. Data stored for each account includes account number, description, budget amount, current month expenses, and year-to-date expenses. Requires posting only once per month. Helps balance checkbook, too!
Copies: **Just released**
Price: **$15**
Includes: Cassette (300 baud Kansas City std), user manual with complete BASIC listings, operating instructions, and sample runs.
Author: **Bruce Grayson**
Available from:
B. W. Grayson
905 Woodridge Drive
Savannah, Georgia 31410

Name: **BASIC Modification Package**
System: **KIM** expanded to run Microsoft-9 digit KIM BASIC
Memory: **Locations DD to E0 and 200 to 2E4** used in addition to locations in unmodified program.
Language: **Machine**
Hardware: **None additional.** Optionally supports a terminal with x-on/x-off feature.
Description: Enhancements and modifications to Microsoft 9-digit KIM BASIC (sold by Johnson Computer). Machine Language patches to original program. BASIC and mods can be loaded with only one tape. Jim Butterfield's Hypertape (and other routines) are relocated to low memory on initialization. SAVE and LOAD at Hypertape speeds. SAVE and LOAD messages improved. SAVE returns to BASIC. Programs with higher line numbers can be appended. This means BASIC subroutines, DATA statements and utility programs (RENUMBER) can be added after program development. Interrupt running programs and listing reliably with ST button. GET (one character or digit) command noted and fixed. Terminals with x-on/x-off feature will load paper or cassetts tapes perfectly. BASIC programs saved on cassette tapes with different initialization conditions can be used interchangably. A 1/10 sec counter can be started, stopped and read under program control. Time and control external events with this jeffrey counter (named after former student and pun intended).
Copies: **> 10**
Price: **$15** check or money order.
Includes: Object code listing, instructions, examples, miscellaneous information and help from the author (by correspondence).
Author: **Harvey B. Herman**
Available from:
Harvey B. Herman
2512 Berkley Place
Greensboro NC 27403

Name: **PET Library**
System: **PET**
Memory: **8K**
Language: **Basic, some Assembler**
Hardware: **No Special**
Description: A variety of PET programs including games, educational, music, astronomy, financial, and many others.
Copies: **100+**
Price **$2.50** first program **$1.50 each additional.**
Includes: **Cassette & Postage**
Order Info.: Send Business envelope and postage for complete list of programs available.
Author: **Russell Grokett**
Available from:
PET Library
401 Monument Rd. #177
Jacksonville, FLA 32211

Name: **LIFE for the KIM-1**
System: **KIM-1** with an **XITEX** VIDEO BOARD.
Memory: **2K** ($2000-$2800 plus 30 bytes on page zero.)
Language: **Assembler**
Description: This program will play Conway's game of LIFE. The program will plant one living cell in mid-screen, and then ask for coordinates, measured from the center, for other living cells. A generation takes about 1/6 second for every birth and death. The program may be patched to accommodate other video boards.
Copies: **Just released.**
Price: **$2.00** for description and listing.
$5.00 for object tape on cassette in HYPERTAPE format.
Author: **Theodore E. Bridge**
Available from:
Theodore E. Bridge
54 Williamsburg Dr.
Springfield, MA 01108

6502 INFORMATION RESOURCES UPDATED

A list of regular publications which have material of interest to 6502 users.

William R. Dial
438 Roslyn Ave.
Akron, OH 44320

Did you ever wonder just what magazines were the richest sources of information on the 6502 microprocessor, 6502-based microcomputers, accessory hardware and software? For several years this writer has been assembling a bibliography 6502 references related to hobby computers and small business systems. The accompanying list of magazines has been compiled from this bibliography. At the top of the list are several publications which specialize in 6502-related subjects. An attempt has been made to give up-to-date addresses and subscription rates for the magazines cited. Subscription rates are for U.S. Other countries normally are higher.

MICRO
$15.00 per year
MICRO
P.O. Box 6502
Chelmsford, MA 01824

6502 USER NOTES
$13.00 per 6 issues
Eric Rehnke
P.O. Box 33093
Royalton, OH 44133

OHIO SCIENTIFIC — SMALL SYSTEMS JOURNAL
$6.00 per year (6 issues)
Ohio Scientific
1333 S. Chillicothe Rd.
Aurora, OH 44202

PET GAZETTE
Free bi-monthly (Contributions Accepted)
Microcomputer Resource Center
1929 Northport Drive, Room 6
Madison, WI 53704

Robert Purser's REFERENCE LIST OF COMPUTER CASSETTES
Nov. 1978 $2.00/Feb 1979 $4.00
Robert Purser
P.O. Box 466
El Dorado, CA 95623

THE PAPER (PET)
$15.00 per year (10 issues)
The PAPER
P.O. Box 43
Audubon, PA 19407

THE CIDER PRESS (APPLE)
Scot Kamins
Box 4816
San Francisco, CA 94101

STEMS FROM APPLE
Ken Hoggatt
APPLE PORTLAND PROGRAM LIBRARY EXCHANGE
9195 SW El Rose Court
Tigard, OR 97223

APPLE SEED
Bill Hyde
The Computer Shop
6812 San Pedro
San Antonio, TX 78216

KILOBAUD/MICROCOMPUTING
$18.00 per year
Kilobaud Magazine
Peterborough, NH 03458

BYTE
$18.00 per year
Byte Publications, Inc.
70 Main St.
Peterborough, NH 03458

DR. DOBB'S JOURNAL
$15.00 per year (10 issues)
People's Computer Co.
Box E
1263 El Camino Real
Menlo Park, CA 94025

ON-LINE
$3.75 per year (18 issues)
D. H. Beetle
24695 Santa Cruz Hwy
Los Gatos, CA 95030

RECREATIONAL COMPUTING
(formerly PEOPLE'S COMPUTERS)
$10.00 per year (6 issues)
People's Computer Co.
1263 El Camino Real
Box E
Menlo Park, CA 94025

INTERFACE AGE
$18.00 per year
McPheters, Wolfe & Jones
16704 Marquardt Ave.
Cerritos, CA 90701

POPULAR ELECTRONICS
$12.00 per year
Popular Electronics
One Park Ave.
New York, NY 10016

PERSONAL COMPUTING
$14.00 per year
Benwill Publishing Corp.
1050 Commonwealth Ave.
Boston, MA 02215

73 MAGAZINE
$15.00 per year
73, Inc.
Peterborough, NH

CREATIVE COMPUTING
$15.00 per year
Creative Computing
P.O. Box 789-M
Morristown, NJ 07960

SSSC INTERFACE
Southern California Computer Soc.
1702 Ashland
Santa Monica, CA 90405

EDN (Electronic Design News)
$25.00 per year
Cahners Publishing Co.
270 St. Paul St.
Denver, CO 80206

RADIO ELECTRONICS
$8.75 per year
Gernsback Publications, Inc.
200 Park Ave., South
New York, NY 10003

QST
$12.00 per year
American Radio Relay League
225 Main St.
Newington, CT 06111

IEEE Computer
IEEE
345 E. 47th St.
New York, NY 10017

POLYPHONY
$4.00 per year
PAIA Electronics, Inc.
1020 W. Wilshire Blvd.
Oklahoma City, OK 73116

RAINBOW (APPLE)
$15.00 per year
Rick Simpson and Terry Landereau, Editors
P.O. Box 43
Audubon, PA 19407

PET USER NOTES
$5.00 per year (6 or more issues)
PET User Group
P.O. Box 371
Montgomeryville, PA 18936

CONTACT — User Group Newsletter
Gratis to Apple owners
10260 Bandley Drive
Cupertino, CA 95014
(408) 996-1010

SOUTHEASTERN SOFTWARE NEWSLETTER
(APPLE)
10 issues $10.00
George McClelland
Southeastern Software
7270 Culpepper Drive
New Orleans, LA 70126

COMPUTER MUSIC JOURNAL
$14.00 per year (6 issues)
People's Computer Co.
Box E
1010 Doyle St.
Menlo Park, CA 94025

POPULAR COMPUTING
$18.00 per year
Popular Computing
Box 272
Calabasas, CA 91302

MINI-MICRO SYSTEMS
$18.00 per year
Modern Data Service
5 Kane Industrial Drive
Hudson, MA 01749

DIGITAL DESIGN
$20.00 per year
Benwill Publishing Corp.
1050 Commonwealth Ave.
Boston, MA 02215

ELECTRONIC DESIGN
(26 issues per year)
Hayden Publishing Co., Inc.
50 Essex St.
Rochelle Park, NJ 07662

CALL A.P.P.L.E.
$10.00 per year (includes dues)
Apple Puget Sound Program Library Exchange
6708 39th Ave. SW
Seattle, WA 98136

6502 BIBLIOGRAPHY
PART VI

William R. Dial
438 Roslyn Ave.
Akron, OH 44320

361. Bridge, Theodore E. "High Speed Cassette I/O for the KIM-1", DDJ 3 Issue 6 No 26, Pg 24-25, (June/July, 1978). Will load or dump at 12 times the speed of KIM-1. Supplements the MICRO-ADE Editor-Assembler.

362. Baker, Robert "KIMER: A KIM-1 Timer", Byte 3 No 7 Pg 12, (July, 1978). The program converts the KIM-1 into a 24-hr digital clock.

363. Conley, David M. "Roulette on Your PET with Bells and Whistles", Personal Computing 2 No 7 Pg 22-24 (July, 1978). How to add extras in a program for added interest.

364. KIM-1/6502 User Notes, Issue 11, (May, 1978)
Lewart, Cass R. "An LED Provides Visual Indication of Tape Input". An LED allows you to see that the tape recorder is feeding proper signals to KIM.
Rehnke, E. "Hardware Comparison". The editor compares KIMSI vs. KIM-4 as expansion for KIM.
Rehnke, E. "Software Comparison". The editor compares the MOS Technology Assembler/Editor from ARESCO versus the MICRO-ADE Assembler/Disassembler/Editor from Peter Jennings, Toronto.
Edwards, Lew "Skeet Shoot, with Sound". Butterfield's "Skeet Shoot" modified with the Kushnier's phaser sound routine, for KIM.
DeJong, Marvin "Digital Cardiotachtometer". KIM counts heartbeats per minute and displays count while measuring next pulse period.
Rehnke, E. "Book review: 'Programming a Microcomputer: 6502'". Foster Caxton's recent book is highly recommended.
Coppola, Vince "Loan Program in FOCAL". FOCAL-65 is used to figure interest on a loan.
Flacco, Roy "Joystick Interface". A joystick, some hardware, are used to put the Lunar Lander (First Book of KIM) on the face of a Scope.
Kurtz, Bob "Morse Code Reader Program". Use KIM in the hamshack.
Zuber, Jim "Interfacing the SWTPC PR-40 Printer to KIM-1". An easy way to use this low cost printer.
Nelis, Jody "Revision to Battleship Game". Modification to correct a small defect in the original program.

365. People's Computers 7 No 1 (July/Aug, 1978).
Cole, Phyllis "SPOT". Several notes and tips of interest to PET owners.
Cole, Phyllis "Tape Talk". Notes on problems associated with tape I/O on the PET.
Gash, Philip "PLOT". Program plots any single-valued function y(x) on a grid.
Julin, Randall "Video Mixer". A circuit to mix the three video signals put out by the PET's IEEE 488-bus.
Bueck/Jenkins "PETting a DIABLO". How to make PET write using a Diablo daisy wheel printer.

366. Harr, Robt. Jr. and Poss, Gary F. "TV Pattern Generator", Interface Age 3 Issue 8 Pg 80-82; 160, (Aug, 1978). Pattern generator in graphics for the Apple II monitor.

367. Personal Computing 2 No 8 (Aug, 1978).
Maloof, Darryl M. "PET Strings" (letter to Editor). Note on changing a character string to numeric values and vice-versa.
Connors, Bob "PET Strings" (letter to Editor). More on changing character strings to numeric values.
Bueck/Jenkins "Talking PET" (letter to the Editor). Notes on the interfacing of a Diablo daisy wheel printer with PET through the PET ADA device.

368. Lasher, Dana "The Kalculating KIM-1", 73 Magazine, No 215 Pg 100-104 (Aug, 1978). Calculator versatility for any KIM is provided by interfacing a calculator chip and a scanning routine with KIM.

369. OSI-Small Systems Journal 2 No 2 (Mar/Apr, 1978).
Anon. "The 542 Polled Keyboard Interface". Polled keyboards have many advantages over standard ASCII keyboards.
Anon. "Basic and Machine Code Interfaces". This is the first in a series of articles on BASIC and machine code.
Anon. "Using the Model 22 OKIDATA Printer". A quick and dirty way to use those special font and scroll commands of the Model 22 OKIDATA Printer.

370. Dr. Dobbs Journal 3 Issue 7 No 27 (Aug, 1978).
Moser, Carl "Fast Cassette Interface for the 6502". Record and load at 1600 baud.
Meyer, Bennett "Yet Another 6502 Disassembler Fix". Changes to correct a number of errors in the five digit codes used for deciphering the instructions in the BASIC language disassembler published earlier in DDJ 3 No 1.
Anon. "Apple Users Can Access Dow Jones Information Service". With a telephone link-up, Apple II users can dial Dow Jones Information Service.

371. Kilobaud Issue 21 (Sept, 1978).
Wells, Ralph "Trouble Shooters' Corner". Another chapter in the saga of the compatibility of the Apple II with a VIA/PIA. See EDN May 20,1978; MICRO Issue 5, Pg 18, June/July, 1978.
Tenny, Ralph "Troubleshooters' Guide". Useful suggestions for those tackling repair and interfacing problems.
Young, George "Do-It-All Expansion Board for KIM". How to make an expansion board, expansion power supply, new enclosure, etc., for your KIM-1.
Ketchum, Don "KIM Organ". Play tunes directly from the KIM keyboard.
Grina, James "Super Cheap 2708 Programmer". An easy-to-build PROM programmer driven by the KIM-1.

372. Conway, John "Glitches Can Turn Your Simple Interface Task into a Nightmare". Difficulties in using an Apple II with a PIA in an I/O interface, apparently caused by a clock signal arriving a little early.

373. Notley, M. Garth "Plugging the KIM-2 Gap". Byte 3 No 9 Pg 123 (Sept, 1978). How to map the KIM-1 address range of 0400 to 13FF into a KIM-2 address range of 1000 to 1FFF.

374. Turner, Bill and Warren, Carl "How to Load Floppy ROM No 5", Interface Age 3 No 9 Pg 60-61 (Sept, 1978). Side No 1 is in Apple II format at 1200 baud, "The Automated Dress Pattern".

375. Smith, Wm. V.R. III "The Automated Dress Pattern for the Apple II". Interface Age 3 No 9 Pg 76-81 (Sept, 1978). A McCalls pattern is the basis for the program and accompanying Floppy ROM.

376. MICRO Issue 6 (Aug/Sept, 1978).
Husbands, Charles R. "Design of a PET/TTY Interface". Describes the hardware interface and software to use the ASR 33 Teletype as a printing facility for the PET.
Faraday, Michael "Shaping Up Your Apple". Information on using Apple II's High Resolution Graphics.
Eliason, Andrew H. "Apple II Starwars Theme". Disassembler listing of theme from Star Wars.
Bishop, Robert J. "Apple PI". How to calculate PI to 1000 places on your Apple II.
McCann, Michael J. "A Simple 6502 Assembler for the PET". Learn to use Machine language with this assembler.
Rowe, Mike "The Micro Software Catalog: III". Software listing for 6502 systems.
Gaspar, Albert "A Debugging Aid for the KIM-1". A program designed to assist the user in debugging and manipulating programs.
DeJong, Marvin L. "6502 Interfacing for Beginners: Address Decoding II". Good tutorial article.
Suitor, Richard F. "Brown and White and Colored All Over". Discussion of the colors in the Apple and their relation to each other and the color numbers.
Witt, James R. "Programming a Micro-Computer: 6502 by Caxton Foster". More accolades for this fine book.
Merritt, Cal E. "PET Composite Video Output". How to get video output for additional monitors.
Quosig, Karl E. "Power from the PET". How to tap the unregulated 8v and regulate to 5v.
Suitor, Richard F. "Apple Integer BASIC Subroutine Pack and Load". Loading assembly language programs with a BASIC program.
Creighton, Gary A. "A Partial List of PET Scratch Pad Memory". Tabulation of a number of important addresses.

377. Corbett, C. "A Mighty MICROMITE". Personal Computer World 1 No 4 Pg 12 (Aug, 1978). Descriptive article on the KIM-1 for the European and British readers.

378. Coll, John and Sweeten, Charles "Colour is an Apple II". Personal Computing World 1 No 4 Pg 50-55 (Aug, 1978). Description of the Apple II.

379. North, Steve "PET Cassettes from Peninsula School". Creative Computing 4 No 5 Pg 68 (Sept/Oct, 1978). A number of programs written in PILOT, a language designed for CAI dialog applications. This requires a program to interpret PILOT in Basic.

380. Gordon, H. T. "Use of NOPCODES as Executable Labels", Dr Dobb's Journal 3, Issue 8 No 28 pg 29 (Sept., 1978). Discusses the use of nopcodes in 650X devices. Classifies these as monops (6 listed) and binopcodes (5 listed) and trinops (8 listed).

381. Swank, Joel "A Programmable IC Tester for KIM", DDJ 3 Issue 8 no 28 pg 33 (Sept., 1978) With a 6820 PIA, some 7404 buffers and a ZIF 16 pin socket together with a program KIM-1 can test IC's.

382. People's Computers 7 No 2 (Sept./Oct., 1978)
Zimmermann, Mark "Snooping With Your PET". A sophisticated Guide to PEEKing and POKEing around in PET.
Gaines, John "Apple Math". A math program for the Apple II.
Cole, Phyllis "SPOT" (Tips for PET trainers) - Discussion of the slow documentation for the PET. Program for lining up dollars/cents tabulations. PET listing conventions.

383. Conway, John "A Tape-to-Microcomputer-Hardware Interface Requires a Wealth of Micro-techniques". EDN 23 No 6 pg 101-110 (March 20, 1978). EDN project Indecomp tape interface hardware. Also the first hint of problems in interfacing Apple II with a PIA.

384. Hemenway, Jack E. "Add Floppies to Your Microcomputer to Form a Real Microcomputer System". EDN 23 No 12 pg 98-107 (June 20, 1978) re: May 20 EDN problems with tape hardware interfaces for Apple II. AIso, EDN 23 discusses disk interface hardware. Final test to come.

385. Kilobaud Issue 23 (Oct., 1978)
Trageser, Jim "Budget System with KIM". How to expand your KIM system with ASCII keyboard, TVT-6 and associated software.
Ngai, Philip "Build a One-Chip Single Stepper". A debugging aid for home-brewed 6502 systems.
Kurtz, Robert L. "World of the Brass Pounders: Receive Morse Code the Easy Way". This Morse code reader is a good example of how the microcomputer can serve the radio amateur. Uses KIM-1.
Borland, D. "Financier/Mortgage with Prepayment". A pair of PET programs from Kilobaud's Instant Software line.
Beymer, Easton "Universal Number Converter". A program in PET BASIC for converting from one number base to another. Not only hex, decimal, octal and binary, but others.
Grossman,,Rick "Do It with a KIMSI". Use S-100 boards with the KIM. An evaluation.
Bishop, Robert J. "The Remarkable Apple II". A description and evaluation of the Apple II by a veteran 6502 programmer.

386. Palenik, Les "Formatting Dollars and Cents". Byte 3 No 10 pg 68 (Oct., 1978) This program for the PET rounds the monetary amount to the nearest cent and lines up the decimal points.

387. Bishop, Robert J. "Maze" Byte 3 No 10 pg 136-138 (Oct., 1978). This novelty program generates and displays a different maze about once a minute.

388. 6502 User Notes 12 (Nov., 1978)
Flacco, Roy "Scope Lunar Lander--final installment". The last part of a series incorporating graphics into the KIM-1 Lunar lander program.
Allen, Michael "TVT6 Etch-A-Sketch". A sketch routine for the KIM/TVT6 said to overcome the limitations imposed by a snowy screen during program execution.
Hooper, Philip K. "TVT6" - Some cheap, easy and helpful TVT6 hardware modifications.
Clem, D. "Expansion Decoding". One possible configuration of expansion decoding for KIM is said to be designed with TVT6 in mind.
Lewart, Cass and Lewart, Dan "TVT6 Remarks". Notes on improving the operation of the KIM/TVT6 system.

6502 User Notes 12 (Nov., 1978) con't.

Kushnier, Ronald "Notes on the TVT6". Information on memory expansion, the TVT6/KIM as a terminal, assembly of the TVT 6, the PVI-1K kit, KIM modification and use of the TVT6 with the Radio Shack keyboard.

Brachman, Michael "Suggestions for Running the TVT6". Includes a slight software mod to display pages 02,03,00 consecutively filling the whole screen (24 line x 64 character display.

Anon, "FOCAL" Staff evaluation of FOCAL for the 6502.

Latham, Don J. "Letter to the Editor". Comments on tape storage problems, KIMSI, XITEX, TVT6,Microsoft Basic, FCL-65, etc.

Rehnke, Eric "The Cheap Video Cookbook" - a book review. Very favorable review of Don Lancaster's latest book.

Martin, Timothy "KIM Interval Timers". Useful precautions in using the KIM interval timers are given.

389. MICRO, No 7 (Oct./Nov., 1978)

Auricchio, Rick "Breaker:An Apple II Debugging Aid". BREAKER is a software routine to manage Breakpoints, correctly resuming the user program after hitting a breakpoint

Watson, Allen III "MOS 16K RAM for the Apple II. Speed codes used by 16K Dynamic RAM manufacturers. The author advises against using 300 ns access time chips.

Creighton, Gary A. "PET Update". Discussion of the RND function, USR, Machine Language Storing in Basic, Save and Load, etc.

De Jong, Marvin L. "6502 Interfacing for Beginners; The Control Signals". The latest article in this series discusses the theororetical basis and progresses to hardware and a program for experimenting with control signals.

Shryock, William H., Jr. "Improved Star Battle Sound Effects". Further improvements based on the original article by Andrew H. Eliason in Issue no 6 of MICRO.

Green, J.S. "650X Opcode Sequence Matcher". A program that correlates and points to parallel sequences of opcodes, comparing the two sets and displaying the differences.

McCann, Michael J. "A Memory Test Program for the PET". Program is written in Commodore BASIC and occupies the lowest 4K of memory.

Rowe, Mike (Micro Staff) "The MICRO Software Catalog: IV" Ten more programs are reviewed in this continuing series.

Schwartz, Marc "Apple Calls and Hex-Decimal Conversion". How to access machine language routines by Calls and how to use Apple in helping make the hex-decimal conversion.

Dial, William R. "6502 Bibliography: Part VI". The 6502 literature continues to expand.

Dial, William R. "6502 Information Resources". A list of the magazines used in compiling the 6502 Bibliography and their subscription prices.

Powlette, Joseph L. and Wright, Charles T. "KIM-1 a Digital Voltmeter". Hardware and Software to convert the KIM-1 to use as a digital readout voltmeter.

Miller, Fred "Cassette Tape Controller". Control two tape units with your KIM.

Eliason, Andrew H. "Apple II High Resolution Graphics Memory Organization". A useful contribution toward better understanding of the Apple II HIRES Graphics.

DeJong, Marvin L., Riverside Electronics Design's KEM and MVM-1024: A User's Evaluation". A generally favorable evaluation of the KEM expansion board for KIM and the keyboard/video monitor board.

Sullivan, Chris "A Digital Clock Program for the SYM-1". This 24-hour clock program provides a good way for the new SYM owner to become familiar with the monitor subroutines.

Herman, Harvey B. "Peeking at PET's Basic". The PEEK function is used to look at the BASIC itself.

Tepperman, Barry "Kimbase". A program to convert from almost any number system to another. Contains many useful subroutines for multiplying, dividing and other uses.

390. Anon, "Tone Routine for Apple II". Southeastern Software Newsletter Issue No 3 pg 6 (Oct., 1978) The tone routine used with Apple Integer Basic can be used with Applesoft but must be relocated.

391. Haller, George "Storing and Recoving Data in Applesoft II". Southeastern Software Newsletter Issue No 2 pg 4 (Sept, 1978) Program for storing and recoving data in Applesoft II.

392. Call - APPLE 1 No 6

Williams, Don "Key Klicker Routine". A machine language program to provide a click each time your silent keyboard is punched

Anon, "Routine to Find Page Length". A routine to fill a page with repetitive material or to determine the length of a screen page of print statements.

Anon, "Printer Driver Fixes". A short program to prevent problems when using the Apple with a printer with more than 40 columns.

Anon, "Apple II Mini-Assembler". Discussion of the Apple II mini-assembler.

Aldrich, Darrell "Use of Color Mask Byte in HIRES". Brief description of this important aspect of Apple HIRES.

Anon, "Memory Map-Apple II with Applesoft Basic Loaded". Convenient tabulation of memory positions.

Anon, "List of Handy Calls". Sixteen calls are listed for the Apple II.

Apple Computer Staff "System Monitor". Discussion of how to get the most out of your Apple Monitor.

Huelsdonk, Bob "Memory Test". A test for Apple memory by loading each location with 55, testing, loading with AA and testing. Offending address will be shown.

393. Call - APPLE 1 No 7 (August, 1978)

Golding, Val J. "A Disk Utility Program". A program to record a group of programs on tape from a disk.

Backman, J. A. "Poor Man's HEX-DECIMAL-HEX Converter". With this table and a scratch-pad, conversions are a snap.

Thyng, Mike "Basic File Handling". Discussion of the actual commands necessary to get data to and from diskettes using the Apple with a PERSCI disk drive.

Apple Computer Staff "System Monitor". Cassette I/O's, Memory Move and Verify, Debugging Aids, Single Stepping, Tracing, using the Apple II monitor.

Anon "Applesoft Zero Page Usage". Explanation of the functions residing in page zero of Apple II.

Huelsdonk, Bob "Routine to Print Free Bytes". Routine for Apple II with less than 32K memory.

Huelsdonk, Bob "A Patch for Double Loops". Discussion of precautions to use with with double loops in Applesoft on the APPLE II.

Apple Computer Staff "Loading Machine Language as Part of a Basic Program". Reprinted from Contact No 1, May 1978. Provides a way to include a machine language program within a Basic Program.

394. Call - APPLE 1 No 8 (Sept., 1978)

Aldrich, Ron "Convert". Program Loads Integer Basic Program from Disk, saves to a text file on disk, then executes that file in Applesoft II.

Thyng, Mike "Arrays". A description of the use of arrays with the Apple.

Chapman, Dan "Video Display Organization". A program to demonstrate the Video Display organization.

Anon "Routine to Save an Array". (reprinted from Apple Stems Vol 1 No 2 July, 1978) A routine to save both integer or floating point real numbers in an Applesoft II array.

Lam, S.H. "Monitor Commands from Basic." This routine allows execution of Apple Monitor commands from Basic with return to Basic.

Williams, Don "Linkage Routines for the Apple II Integer Basic Floating Point Package". A discussion of the Apple II ROM routines.

Hill, Alan G. "Return to TEXT from Graphics". A handy routine which permits the use of Control Y to return to Text from Apple Graphics.

Anon "Integral Data IP 125-225 Driver". A slight modification of the Apple Red Book teletype routine for use with the Integral Data printer.

Huelsdonk, Bob "Printer Driver Fixes". Protocol to use a printer with more than 40 columns with the Apple.

395. Call - APPLE 1 No 9 (Oct., 1978)

Cook, John B. "Applesoft Tone Routines". Relocation of the tone routines is necessary for use with Applesoft II on the Apple.

Scott, Michael M. "A Brief History of Apple". An interesting account by the President of Apple Computer Co.

Call - APPLE 1 No 9 (Oct., 1978) cont.

Anon, "Some Basic Entry Points". Various Call and JSR functions for Apple basic.
Huelsdonk, Bob "Sample File Handler". A program that demonstrates and will establish files for data handling, using the DOS on Apple.
Golding, Val and Williams, Don "Apple II Integer Basic: Interpretation of Memory". Tabular listing of pointers and tokens for Apple II Integer Basic.
Golding, Val "Applesoft II Tokens". Memory tabulation for tokens and pointers of Applesoft II Basic for the Apple.

396. PET Gazette 1 No 5 (Aug./Sept., 1978)

Anon, "PET Standards". Standards are suggested for writing PET programs, graphics, music listings, etc.
Louder, Mike "Dynamic Keyboard". Discussion of methods to change lines in Basic programs while the PET is running.
Richter, Mike "Data Files". Recommended procedures in preparing tapes for exchange.
Anon, "PET Memory Map". Map reprinted from the PET Paper.
Staebell, Jon "PET Hints". Miscellaneous hints for PET owners.
Modeen, Roger L. "Easy Auto Answer/Originate Modem". Modem for the PET.
Clark, Ken "Proposed Temporary Local Standard for Low Speed Data Exchange by Modem". Protocals for data exchange.
Cumberton, Dennis "File Manager". A program for reliable data reading and writing with the PET.
Bendoritis, Bill "Renumber". A renumbering program for the PET.

397. Babcock, Robert E. "1C Tester Using the KIM-1"
Ham Radio 11 No 11 pg 74-76 (Nov., 1978)
Test the 7400 series IC's using the KIM-1, a minimum of hardware and tables of parameters tucked away in memory.

398. Purser, Robert "Software List"
Robert Purser's Reference List of Computer Cassettes, Edition 3, August, 1978 (P.O. Box 466, El Dorado, CA 95623
A very complete listing of available Cassette software for the PET and Apple II.

399. Lilie, Paul A. "Look What Followed Me Home!"
73 Magazine No 218 pg 142-147 (Nov., 1978)
A description of the PET.

400. Creason, Sam "The Micro Maestro!"
73 Magazine No. 218 pg 150-166 (Nov., 1978)
Sound generation and waveform control with the 6502.

401. Akingbehin, Kiumi "LEDIP, A KIM/6502 Test Editor"
Dr. Dobb's Journal 3 Issue 9 No 29 pg 4-12 (Oct., 1978)
Here is an expandable program for creating text and source code.

402. Tepperman, Barry "Comments on KIM Cassette Program"
Dr. Dobb's Journal 3 Issue 9 No 29 pg 41 (Oct., 1978)
Points out that the relatively slow speed of the KIM cassette program has led to the publication of several high-speed load/dump programs.

403. Firebaugh, Morris; Johnson, Luther and Stone, William "A Feast of Microcomputers"
Personal Computing 2 No 11 pg 60-70 (Nov., 1978)
The Authors evaluated a wide range of microcomputers to pick the best ones for teaching science students. Includes several 6502 micros.

404. Creative Computing 4 No 6 [Nov./Dec., 1978]
Foote, Gary A. "Apple Speed"
A comparison of several programs for sorting a group of 1000 words showed several BASIC programs to require 600 to 650 seconds on the Apple II while usir the same sort in Sweet-16 required only 158 seconds. The same sort in 6502 assembler required only 3 seconds.

Ahl, David "Random Ramblings"
Commodore plans to make an electronic che s game based on the 6504 chip of MOS Technology.

Yob, Gregory "Personal Electronic Transactions"
A column on the PET with software references, hints on operating, etc.

Milewski, Richard A. "Apple-Cart"
A column on the Apple II with software reviews.

Butterfield, Jim "Games–Not Just For Fun"
The author urges micro users to have fun with their computers; don't be ashamed of games and recreational programs. Creating programs is in itself a highly instructive experience.

405. Dinnell, Rob C. "Graphics Program"
Interface Age 3 Issue 11 pg 14 (Nov., 1978)
Graphics program for the Apple II.

406. Schumacher, Ernst "Sweets for KIM Spurned"
Byte 3 No 11 pg 146 (Nov., 1978)
A fix for a bug in the Sweets for KIM program, Byte Feb., 1978 pg 62.

407. Creative Computing 4 No 5 [Sept./Oct., 1978]

Ahl, D.H. "Personal Computing: The size of the Market"
Out of a total market for personal computers sold in the first three years of 150,000 units, PET is said to account for 15000, TRS-80 for 8000 to 20000 and Apple for 25000 units. All others together account for 75000 to 100000.

Ahl, D.H. "The Home Computer: A Tool Not A Toy"
An interview with Mike Scott, President, Apple Computer.

Ahl, D.H. "Home Computers: The Name of the Game is Peripherals"
An interview with Chuck Peddle, designer of the Commodore PET.

Ahl, D.H. "Reliability and Mass Production"
The most frequent computer problems over all manufacturers including the 6502 types fall into two categories: (1) Cassette recorder, mostly head misalignment and (2) overheating errors after running a while.

North, Steve "PET Cassettes from Peninsula School"
A review of software available from the Peninsula School of Monlo Park, CA.

408. Anon, "12-Test Benchmark Study Results Show How Three Microprocessors Stack Up"

EDN 22 No 21 pg 19 (Nov. 20, 1978)
Once again the 6502 is shown to be substantially faster than the 8080 or 6800, as well as using less memory.

409. Anon, "Project Indecomp--EDN Builds a computer System"

EDN 22 No 21 pg 221-233 (Nov. 20, 1978)
Outlines the beginning of this project that was to provide material for a number of articles to follow, principally on methods of interfacing to a tape deck.

410 Schreirer, Paul G. "Low-Cost System Requirements Multiply Interface Headaches"

EDN 23 No 3 pg 39-44 (Feb. 5, 1978)
Interconnecting a cassette system to Indecomp proved tough due to strong chip-discrimination against the 6502 in this 8080/Z80 world.

411 Call - Apple 1 No 10 [Nov./Dec., 1978]

Anon, "Use of Apple II Color Graphics in Assembly Language"
Tutorial article on graphics

Jackson, Gene "Checkbook Changes for Disk"
Modifications for this popular program for the Apple Disk.

Paulson, Steve, "Using Game-Paddle Buttons"
How to change keyboard control over to the paddle buttons.

Anon, "& Now, the Further Adventures of the Mysterious Ampersand."
Continued from last month--more on the functions of the character "&", in Applesoft routines.

Wigginton, R. "Simple Tones--A Demonstration for Extensions to Applesoft II."
Simple tone program for Applesoft II inside the Applesoft Program.

Finn, Jeffrey K. "Apple-Sharing" Part I of II
Part I of a tutorial article on time sharing and the Apple.

Thyng, Mike "Apple Mash"
This issue discusses how and why the DIMensioned statement works, Alpha String arrays, Integar and Floating Point Arrays, etc.

Anon, "Peeks, Pokes and Calls"
A discussion of the utility of these very useful tools.

Thyng, Mike "Apple Source"
Question and answer session with Mike Scott, President of Apple Computer and Randy Wigginton of Apple.

Golding, Val J. "Identifying Binary Disk Programs"
Ways to help you save and identify machine language programs on disk.

Anon, "Resurecting a Dead FP Program."
Methods to help you retrieve an Applesoft II program that has blown up while you were working on it.

412. Southeastern Software Newsletter Issue No 4 [Nov., 1978]

Anon, "Hires Graphics"
Examples of how to program in Hires machine language. Also includes a program in Applesoft II called Random Walk

Anon, "How to Use "Quotation" Marks in a Print Statement."
Tricky in Applesoft II to make the quote marks print.

Hartley, Tim "How Memory is interpreted in Integar Basic"
A program to list the tokens used in Integar Basic.

Banks, Guil "Programs for Disk"
Two programs are given. EXEC GEN and READ FILE.

Anon, "Applesoft in Firmware"
A discussion of the use of the Applesoft II ROM card.

413. Carpenter, C.R. [Chuck] "Pilot for the Apple"

People's Computers 7 No 3 pg 4 (Nov./Dec., 1978)
An extended version of PILOT for the Apple Disc II is being written.

414 Cole, Phyllis "SPOT"

People's Computers 7 No 3 pg 48-51 (Nov./Dec., 1978)
Hints on using the Commodore PET include tips for loading balky tapes from the cassette, adding an auxilliary keyboard, and review of new software.

415. Greenberg, Gary "Phone Directory"

Personal Computing 2 No 12 pg 34-35 (December, 1978)
A PET program provides rapid access to a phone number without a random access filing system.

416. Zimmermann, Mark "Assembler for the PET"

Personal Computing 2 No 12 pg 42-45 (December, 1978)
This BASIC program lets you write in Assembly Language.

417. Gable, G.H. "Zapper--A Computer Driven EROM Programmer"

Byte 3 no 12 pg 100-106 (December, 1978)
The Zapper is a Erom programmer using a KIM-1 as driver for the Zapper.

418. Watson, Allen, III 430 Lakeview Way, Redwood City, CA 94062

Byte 3 No 12 pg 208 (December, 1978)
Notes on minimizing TV interference by the Apple II.

419. Lantz, Kim H. "RTTY with the KIM"

73 Magazine Issue 219 pg 170-173 (December, 1978)
This article goes a step further and uses the KIM to deliver the RTTY to the HAL terminal.

420. Anon. "Bringing up the New Disk"

Southeastern Software Newsletter Issue No 5, Pg 2 (Dec., 1978)
Hints and Kinks on putting that newly delivered Apple Disk to work. Making duplicate masters, creating random files, reading back files, transferring programs from one disk to another for backup, etc.

421. PURSER, ROBERT E. "Reference List of TRS-80, PET and APPLE II Computer Cassettes"

Edition 4, November, 1978 (P.O. Box 466, El Dorado, CA 95623

A very complete listing of software for the Apple II and PET is given. A few software reviews are given.

422. MICRO No 8 (Dec., 1978-Jan., 1979)

De Jong, Marvin L. "6502 Interfacing for Beginners: Buffering the Busses"

The author continues his series of tutorial articles discussing the need for buffers, types of buffer chips and some ex-experiments and an application.

Anon. "Microbes"

An entire section of code from "Breaker: An Apple II Debugging Aid" MICRO NO 7 pg 5 was omitted and is given in this correction. Also a correction for Husband's "Design of a PET TTY Interface" Micro No. 6 pg 5.

Suitor, Richard F. "Life for your Apple"

A new version of LIFE has the generation calculations in assembly language to speed the program.

Reich, Dr. L.S. "Computer-Determined Kinetic Parameters in Thermal Analysis"

A program for the quantitative estimation of kinetic parameters for the material being degraded such as activation energy and reaction order. Uses Apple II.

Christensen, Alan K. "Continuous Motion Graphics or How to Fake a Joystick with the PET"

Basic supported routines are too slow to allow smooth movement. Action is enhanced by direct access of screen and keyboard.

Powlette, Joseph L. and Jeffery, Donald C. "Storage Scope Revisited"

With the hardware changes suggested in this article the performance of DeJong's program to transform an ordinary oscilloscope to a storage scope gives results approaching those of a commercial unit.

Auricchio, Rick, "An Apple II Program Relocator"

A program to move an Assembly language program to another part of memory. Changes all absolute references within the program.

Gieryic, John, "SYM-1 Tape Directory"

Program to allow the SYM owner to examine his cassette tape to find what information is there.

Anon. "The Best of MICRO Volume 1"

A book containing most of the articles that were published in MICRO Volume 1.

Butterfield, Jim "Inside PET Basic"

Two new programs for PET. FIND will search a PET BASIC program for a particular data string that will list the lines containing the string. RESEQUENCE will renumber your program fixing up GOTO's and other functions.

Connolly, M.R. Jr. "An Apple II Page 1 Map"

This article shows a clever method of creating all sorts of nifty effects, title pages, etc., on your Apple.

Dial, Wm R. "6502 Bibliography, Part VII"

Some 88 more references to the growing 6502 literature.

423. PET Gazette (Oct./Nov., 1978)

Anon. "Software Reviews — PET"

Many reviews of PET software are to be found scattered through this issue of the Gazette. Also review of many new hardware items for PET.

Staebel, Jon "PET Hints"

PET Gazette 1 No 6 pg 6, (Oct./Nov., 1978)

A discussion of the timer and built-in clock in the PET, examples of use. How to stimulate a repeat key on the PET.

Barsanian A. "Tape Tips"

PET Gazette 1 No 6 pg 8-9 (Oct./Nov., 1978)

Some sensible tips on using PET tape cassettes, storage, copying. How to locate one program out of many on a tape.

Cumberton, Dennis "Tape Tips"

PET Gazette 1 No 6 pg 9 (Oct./Nov., 1978)

Recommendations on the use of C-30 cassettes stripped down to C-10 equivalent. Comments on brands found satisfactory.

Stone, Mike "Program Overlays"

PET Gazette 1 No 6 pg 11-13 (OCT./Nov., 1978)

Joining two programs.

Anon. "New PET Booklet-PET Communicates with the Outside World"

PET Gazette 1 No 6 pg 15-19 (Oct./Nov., 1978)

Summary of the important information released including pinout for Parallel User Port, Second Cassette Interface, and Memory Expansion Connector. IEEE Bus Limitations, I/O Commands, I/O operations, Recording techniques, Error Detection, etc., etc.

Lindsay, Len "Kilobaud Column for PET Users"

Hints on programming with your PET, Use of the GET command.

424. Dr. Dobb's Journal 3 Issue 10 No. 30 (Nov./Dec., 1978)

Bridge, Theodore E. "A Curve-Fitting Program Using a Focal Interpreter on the KIM-1"
Focal is used with the KIM in a curve-fitting program.

Swanks, Joel "Tiny GRAFIX for Tiny BASIC"
Grafix is a system for graphic display on a small computer system, including Pittman's Tiny Basic, a SWTPC GT-6144 TV Graphics board, some machine language subroutines and a KIM-1 with 4K of memory.

Oliver, John P. "Astronomy Application for PET FORTH"
Using a newly available language PET-FORTH version 1.0, a PET was used to provide control functions for a telescope.

425. Kilobaud No. 25 (Jan., 1979)

Lindsay, Ien "PET Pourri"
A new column on the PET has sections discussing Accessories, Publications, Software, Programming Hints, and PET Problems. A very helpful series of hints on Cassette recorder maintenance and saving data is included.

Brisson, Dennis "New Products"
6502 products include reviews on weight control/biorythm programs, a telephone cost-control center, the RS-16-HP "universal" interface for PET, a 6502 Assember for PET, a PET Word Processor, etc.

Fuller, Steve "OSI User Group"
The Newton Software Exchange, PO Box 518, Newton Center, MA 02158, is forming a user's group for OSI products, especially the Challenger series.

Anon, "Letters"
This month 6502 letters refer to the November article "Do It with a KIMSI", the September Article "Super Cheap 2708 programmer, etc.

Lang, George E. "u-Panel"
See the reaction of every register of your microprocessor as you single step your KIM through a program.

Ketchum, Don "Display Your PET"
Watch the Monitor screen as all 316 PET characters appear on the Screen

Carpenter, Charles R. "SHHH... People are Sleeping"
The Telpar PS-40-3C-1 serves as a quiet and economical substitute for a noisy and expensive teletype.

Yob, Gregory "PET Techniques Explained"
Supplementing information from Commodore, this article gives information on cassette files.

426. Calculators/Computers Magazine 2 Issue 7 (Nov./Dec., 1978)

Costello, Scott H. "Hilo–A Number-Guessing Program that Illustrates Several Math Concepts"
Modifications for the program to run on several different computers, including PET are given. A number of variations are suggested.

Albrecht, Bob and Albrecht, Karl "PET BASIC for Parents and Teachers"
An explanation of many of the keys on the PET keyboard.

427. Dr. Dobb's Journal 4 Issue 1 Number 31 (Jan., 1979)

Seiler, Bill "PET BASIC Renumber"
A program to put your line numbers in a more ordinary list.

Moser, Carl W. "Add a Trap Vector for Unimplemented 6502 Opcodes"
Ideas on how to provide hardware and a program to ferret out those hidden opcodes

Aresco, P.O. Box 43, Audubon, PA 19407 "6K Assembler/Text Editor for Apple II"
A 6K machine language for the Apple II.

Terc Services, 575 Technology Sq., Cambridge MA 02139 "KIM-1 Interface Set"
Permits easy access to the I/O ports on the KIM.

428. Byte 4 No. 1 (Jan., 1979)

Helmers, Carl "Pascal Progress"
The University of Calif at San Diego plans to make the UCSD Pascal system available on Apple II computer early in 1979.

PRS The Program of the MonthCorporation, 257 Central Park West New York, N.Y. 10024
A2FP is a Plotting Program for Apple II which plots 2-dimensional functions in high resolution graphics.

Leff, Alan A. and Boos, D.L. "A Timely Modification to KIMER"
Modification of the Baker program "Kimer: A KIM-1 Timer" Byte, July 1978, pg 12 to allow it to run as 12 hour clock.

429. Recreational Computing 7 No 4 Iss 37 (Jan./Feb., 1979)

Carpenter, Churck "APPLE II Easy I/O Sensing and Control". I/O control using the Apple II game connector.

Wells, Arthur Jr. "Some New Uses for Apple II". Debugging PONG, use of Modem, etc.

Shanis, Daniel"Breaking Trail in Alaska with Apple II". A project using 32K Apple II computers with diskettes in 9 remote village schools.

Swenson, Carl "Building a HI-RES SHAPE TABLE for the APPLE II". Heres a way to create your favorite shapes.

Saal, Harry "SPOT". Tips for the PET Owners. A machine language tape with two BASIC programs is available from Commodore. Also a manual on communication with the outside world. A PET SERVICE KIT from Commodore includes schematic diagrams and parts lists, a diagnostic jumper connector with diagnostic tapes, etc. Information on the "lost cusor fix". Head Alignment.

430. MICRO No 9 (Feb., 1979)

Reich, Dr. L.S. "Long Distance Interstate Telephone Rates". An Applesoft II program for phone rates which can be modified for PET or OSI computers.

Bullard, GARY J. "The Sieve of Eratosthenes". A prime numbers BASIC program for the PET.

Hertzfeld, Andy "Exploring the Apple II DOS". Useful information for disk users.

DeJong, Marvin L. "6502 Interfacing for Beginners: An ASCII Keyboard Import Port". Shows a system for the KIM with both polled or interrupt methods of service the device.

Tater, Gary L. "Two Short TIM Programs". One program provides a method for communcation with TIM at 1200 BAUD or higher. Another offers a TIM Operating System Menu.

Tripp, Robert M. PhD "Ask the Doctor-Part 1". A comparison of the KIM, SYM and AIM microcomputers.

Watson, Allen "Two APPLE II Assemblers: A comparative Software Review". Advantages and disadvantages of the Microproducts and S-C Assemblers for the Apple II.

Rowe, Mike "THe MICRO Software Catalog: V". Reviews of about one dozer programs for 6502 based systems.

Rittimann, Russell "Expand Your 6502-Based TIM Monitor". A modification of the TIM system to expand the command set so that ROM resident programs or routines can be executed from within TIM.

Dial, Wm R. "6502 Bibliography— Part VIII". The 6502 literature continues to expand.

Sandberg, Gary P how Does 16 Get You 10?". Hexadecimal/Decimal conversions for the Apple.

Herman, Harvey B. "How Does Your ROM Today". Programs and techniques for testing the KIM and PET ROMs.

Bridge, Theodore E. "Life for the KIM-1 and an XITEX Video Board". Program runs on a 16K Kim.

431. CONTACT Newsletter No 4 (Dec. 1979)

Anon "Apples work PIA's". A note to the effect that the problems reported earlier by END magazine on the apparent incompatibility of the Apple with PIA's have been resolved and that EDN now believes this long saga must have had its source in human error. (See EDN Magazine, Sept. 20, 1978)

Anon, "The Colon as a Listing Formatter for Applesoft". How to indent your listings for neatness and easy reading.

Annon "Disk Operating System Notes". Includes Notes on Data Format, Using Random-Length Records, Using Fixed-Length Records, Appending Files, DOS Error Codes, Getting Commas into Applesoft, etc.

432. Dr. Dobb's Journal 4 No 32 Issue 2 (Feb., 1979)

Gordon, H.T. "An Unusual Pseudorandom Number Generator Program". Program for the KIM-1.

Carpenter, Chuck "Reset Adapter". How to avoid accidental loss of programs involving the reset button on the Apple II

Prigot, Jonathan M. "Loading Kim's Cassettes". How to load OSI cassettes into the KIM.

433. Byte 4 No 2 (Feb., 1979)

Libes, Sol "Byte News". Atari has two new 6502 based computers. According to the latest sales reports, more 6502 microprocessors are being manufactured than any other uP. Most of the volume goes to high volume game use.

Mathews, Dr. Randall S. "An Apple and the Queens". An Eight Queens program for the Apple.

Raskin, Jef "Unlimited Precision Division". A BASIC program for unlimited integer division.

434. Kilobaud No 26 (Feb., 1979)

Green, Wayne "Publisher's Remarks". A review of OSI's new units the Ip and IIP Challengers.

Lindsay, Len "PET Pourri". Accssories for the PET include a voice input module, a sound output module, Joysticks, a digital plotter, a light pen, and an S-100 adapter for the PET. How to add sund to your PET and sound programming instructions New languages to supplement Basic are PILOT and PETFORTH. New sources of information on the PET are the PET Manual and a manual called PETABLE, as well as a newsletter called Sphinx. Programming tips cover the GET, ON. . .GOSUB, and others. A new wrinkle for recovering programs from faulty tapes is given.

Flogel, Ekkehard "Apple and the PIA". Contradicting the troubles reported by EDN magazine, a board was developed with a PIA 6520 on it to put an Apple II and a KIM together. Programs can be sent from one unit to the other and vice versa.

Price, David "Music, Maestro". The AD8 is a computer-controlled synthesizer system using a 6502 microprocessor and a 6820 I/O port.

Bishop, Robert J. "The Apple Speaks. . .Softly". Apple II Voice digitizer.

435. Calculators/Computers Magazine 3 No 1 (Jan./Feb., 1979)

Day, Jim "High-Resolution Apple Art". Applesoft II program for various shapes.

Albrecht, Bob and Karl "PET BASIC for Parents and Teachers". PET Conventions in a simple snimation program.

436. 73 Magazine No 221 pg 21 (Feb., 1979)

Birman, Paul "Petting". How to find the end of a program on tape when you want to load a new program into your PET.

437. Personal Computing 3 No 2 pg 63-74 (Feb., 1979)

Gerue and McNeil, "Chess Challenger-10 Wins Microchess Tourney". Microchess 2.0, Peter Jennings entry, took fourth place. This is 6502 based.

437. Creative Computing 5 No 1 (Jan., 1979)

Yob, Gregory "Personal Electronic Transactions". New products described are Expandapet memory, PET ROM disassemblies, a useful book on what the PET rom is all about, Some data on the User Port, PET Video Slave display, Exploring PET random numbers, PET sounds and music, etc.

Wells, Ralph "HOw about a 'Counterfeit Cursor' For your PET?". Enables one to use the cursor in games or under better control.

Heuer, Randy "Ohio Scientific Superboard II and Challenger 1P". A review of OSI's new lost cost micro.

Rugg and Feldman "Speed Reading Made Easy...via Your PET". This program turns your computer into a tachnistoscope to teach improved reading habits.

Milewski,Richard A. "Apple-Cart". All about Data Files on diskettes. Simple file accessing statements, Sample serial access programs, and some software reviews.

NOTE: The Bibliography is continued from "The BEST of MICRO Volume 1".